MASTER EN SCIENCES INFORMATIQUES

Systèmes multi-agents orientés objectifs supportant des processus de workflow

Jamart, Sébastien; Jeanpierre, Jonathan

Award date:
2005

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année Académique 2004 - 2005

Systèmes multi-agents orientés objectifs supportant des processus de workflow

Sébastien Jamart & Jonathan Jeanpierre

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique.

Remerciements

Nous souhaitons remercier toutes les personnes qui ont contribué à la rédaction de ce mémoire. En particulier, nous remercions Madame Monique Noirhomme, notre promotrice, pour son aide et les conseils précieux dont elle nous a fait profiter.

Nous désirons également remercier Messieurs Simeon Simoff et John Debenham, Professeurs à l'Université des Technologies de Sydney, ainsi que leurs équipes, pour l'accueil qu'ils nous ont réservé et le temps qu'ils nous ont consacré tout au long de notre stage.

En particulier, nous remercions Anton Bogdanovich et Helmut Berger, chercheurs à l'UTS, pour leurs expertises techniques dont nous avons bénéficié ainsi que tous les moments agréables que nous avons partagés avec eux et leurs proches durant notre séjour.

Nous souhaitons également remercier Marie-Anne Debout, Laurence Strodiot, Philippe Jamart et Thomas Matelart pour le temps qu'ils ont consacré à la relecture de ce mémoire.

Enfin, nous remercions tous nos proches, et surtout nos parents, pour le soutien dont ils ont fait preuve pendant nos études et la chance qu'ils nous ont offerte de vivre un stage inoubliable.

Résumé

La programmation orientée agents est un paradigme de programmation relativement récent qui trouve son origine dans la recherche en intelligence artificielle. Les systèmes multi-agents (SMA) sont particulièrement appropriés aux environnements dynamiques, imprévisibles, incertains et où les échecs sont possibles. Certains agents sont dits proactifs : ils se voient attribuer des objectifs, par des utilisateurs ou par le système, puis raisonnent afin de trouver la façon la plus adéquate de les atteindre.

Ce mémoire a pour objectif de mettre en évidence l'efficacité des SMA dans le workflow management (WM). Le WM traite, de façon active, les flux de travail entre les participants d'une organisation. Nous avons étudié les caractéristiques commmunes entre agents et workflows, souligné les lacunes actuelles des applications classiques de WM, montré en quoi ils sont complémentaires et comment les agents peuvent améliorer considérablement les systèmes de WM d'aujourd'hui. Par la suite, nous avons insisté sur l'efficacité des agents proactifs ou orientés objectifs qui répondent bien aux contraintes des environnements des workflows.

Finalement, nous analysons les besoins importants en méthodologies et en modèles de référence qui apparaissent suite à l'adoption des deux technologies précitées, afin de pouvoir développer au mieux des systèmes de workflow management basés sur des systèmes multi-agents proactifs.

Mots-clefs : Système multi-agents, BDI, Gestion de workflow, Agent, Proactif, Orienté objectifs.

Abstract

Agent-oriented programming is quite a recent programming paradigm derived from several pieces of research in the field of artificial intelligence. Multiagent systems (MAS) work well in dynamic, unpredictable and uncertain environments where failures are possible. Some agents are proactive : they receive goals from users or systems and work out the best way to achieve these goals.

This master thesis highlights the efficiency of MAS in workflow management (WM). Workflow management can handle all the workflows between the members of an organisation in a proactive way. We have studied common characteristics of agents and workflows, underlined gaps of classical workflow applications and shown how they are complementary and how agents can improve today's workflow management systems. Furthermore, we insist on efficiency of proactive or goal-directed agents that work well in workflow environments.

Finally, we analyse the important need of methodologies and reference models that appears when combining both technologies in order to develop goal-driven multiagent systems supporting workflow management.

Keywords : Multiagent system, BDI, Workflow management, Agent, Proactive, Goal-driven, Goal-directed.

Table des matières

Table des figures

Liste des tableaux

Introduction

Les organisations actuelles ont pris conscience de l'utilité de la gestion des flux de travail internes plus connue sous le nom de workflow management. Ce dernier est présent partout au sein des entreprises au niveau de leur informatique de gestion. En effet, de nombreuses sociétés utilisent des progiciels tels que des ERP (*Entreprise Ressource Planning*). Ces derniers permettent essentiellement une augmentation de la qualité des services, une amélioration des services aux clients, une augmentation de la productivité, une réduction des coûts ainsi qu'une réduction de la vulnérabilité de l'organisation aux changements environnementaux.

Pourtant, la rigidité des systèmes actuels n'est pas en accord avec les environnements dynamiques, complexes, hétérogènes qui caractérisent les organisations humaines d'aujourd'hui. Les ERP sont décriés pour le contrôle du travail humain qu'ils exécutent dans l'unique souci de rentabilité et pour la taylorisation qu'ils imposent. La codification de l'information et la gestion électronique de documents contribuant à fractionner la chaîne de travail permettent un renouveau des principes tayloriens, non plus au sein des industries mais bien au sein des services.

Nous pensons que les systèmes multi-agents orientés objectifs devraient permettre de résoudre la rigidité des technologies actuelles du workflow management face aux environnements dynamiques, complexes et hétérogènes de nos jours. Ils devraient aussi atténuer le sentiment de contrôle.

Nous avons structuré notre mémoire en quatre parties. La première partie présente l'état de l'art des systèmes multi-agents proactifs. Ainsi, dans le chapitre 1, nous définissons les concepts d'agent, de perception, d'action, d'évènement, d'objectif et de croyance. Nous identifions différents types d'agents que nous classifions ensuite. Nous expliquons notamment les différences entre les agents réactifs et proactifs. Nous constatons que les agents proactifs sont des agents intelligents capables de raisonner puis nous définissons l'architecture abstraite des deux types d'agents précités. Au chapitre 2, nous détaillons le modèle BDI. Ce modèle permet de mettre en pratique un raisonnement orienté objectifs très similaire à celui des êtres humains. Nous définissons les concepts du modèle puis nous nous attardons sur son architecture. Au chapitre 3, nous rappelons les étapes indispensables à la construction de tout logiciel complexe, tel un système multi-agents orientés objectifs, et la qualité requise dans la mise en oeuvre de chacune de ces étapes.

La deuxième partie est pour nous la possibilité d'évoquer les méthodologies et les plates-formes de développement de systèmes multi-agents, respectivement à

travers les chapitres 4 et 5. Au cours du chapitre 4, nous insistons sur la nécessaire existence de méthodologies orientées agents en portant un regard critique sur les différences existant entre le paradigme de programmation objets et les agents que beaucoup considèrent comme une suite logique de ce paradigme. Nous présentons ensuite des travaux de comparaison d'une part, des différentes méthodologies de développement orientées agents (chapitre 4) et d'autre part, des diverses plates-formes couramment citées et utilisées par les plus avertis (chapitre 5). Sur base de ces travaux, nous effectuons et justifions notre choix de méthodologie, Prometheus, développée à la « RMIT University » de Melbourne et notre choix de plate-forme, Jack^{TM}, développée par Agent Oriented Software Pty. Ltd. toujours à Melbourne.

Au cours de la troisième partie, nous nous attachons à démontrer en quoi les systèmes multi-agents orientés objectifs et les workflows sont très complémentaires. En guise de chapitre introductif de cette partie, le chapitre 6 aborde de façon détaillée le concept de workflow management. Les chapitres 7 et 8 sont les pièces maîtresses de ce mémoire. Le chapitre 7 concrétise les espoirs que nous plaçons dans les systèmes multi-agents. Nous y présentons les caractéristiques communes des agents et des workflows, leur évidente complémentarité, les lacunes des techniques de workflow management actuelles et les bénéfices potentiels de l'utilisation d'agents dans des activités de gestion de workflow et les différentes possibilités d'intégration technique de ces deux technologies dans un système unique. Enfin, nous terminons ce chapitre 7 en mettant en évidence le besoin de méthodologies de devéloppement de systèmes combinant les deux technologies précitées et le besoin de modèles de référence sous-jacents. Finalement, nous proposons deux solutions d'adaptation de méthodologies et avançons quelques hypothèses utiles à prendre en compte lors de travaux futurs. Au cours du chapitre 8, le dernier de cette partie, nous revenons sur les agents orientés objectifs. Ces derniers nous semblent bien adaptés à la gestion de processus de workflow management. Il s'agit donc pour nous de compléter le chapitre 7 qui concerne la combinaison « agent-workflow » en insistant sur un type d'agents bien particulier qui montre des qualités indéniables pour être performant dans ce domaine.

La quatrième partie sera consacrée à la réalisation de notre travail. Au chapitre 9, nous présentons les grandes lignes de notre projet à la « University of Technology, Sydney ». Le chapitre 10 présente la première partie de notre projet. Celle-ci nous a permis d'assimiler et de mettre en pratique les concepts des systèmes multi-agents, les étapes d'élaboration de logiciel complexe, la méthodologie Prometheus et la plate-forme de développement Jack que nous avons respectivement détaillés dans les chapitres 1, 3, 4 et 5. Enfin, le chapitre 11 de notre mémoire présente la deuxième partie de notre projet. Lors de celle-ci, nous avons pu appréhender le raisonnement de systèmes multi-agents décrit dans le chapitre 2 et les liens qui existent les workflows et les agents orientés objectifs que nous avons détaillés dans le chapitre 8.

Première partie

Etat de l'art

Chapitre 1

Agents et systèmes multi-agents

Sommaire

1.1 Introduction

La programmation orientée agents est un paradigme de programmation relativement récent qui provient de la recherche en intelligence artificielle. Elle répond à un besoin des systèmes informatiques rationnels en offrant des comportements semblables à ceux des êtres humains. Les systèmes informatiques traditionnels ne peuvent que très difficilement modéliser le comportement humain, et souvent, ces logiciels sont limités, en particulier quand ils doivent fonctionner en temps réel.

1.2 Historique

L'origine des agents informatiques se situe dans la limitation des possibilités qu'offrait l'intelligence artificielle classique. L'évolution des domaines d'application vers l'aide à la décision, la reconnaissance et la compréhension des formes, la conduite des processus industriels, ont montré les limites de l'approche classique de l'intelligence artificielle. En effet, celle-ci s'appuie sur une centralisation de l'expertise au sein d'un système unique.

Durant les années 70, des travaux sur la concurrence et la distribution ont contribué à la naissance d'une nouvelle discipline : l'*Intelligence Artificielle Distribuée*[1] (IAD) [Huh87, GH89]. L'objectif de l'IAD était de remédier aux limitations de l'intelligence artificielle classique en proposant une distribution de l'expertise sur un groupe d'agents devant être capables de travailler et d'agir ensemble dans un environnement commun, et donc de coopérer, de négocier, et de coordonner leurs actions. L'idée est que les problèmes informatiques sont de plus en plus complexes et qu'il est par conséquent nécessaire de décomposer les systèmes en sous-systèmes moins complexes capables de résoudre chacun un sous-problème. La coopération de plusieurs sous-systèmes permettrait de résoudre des problèmes globaux d'une plus grande complexité.

Cependant l'IAD a posé de nouvelles questions. Comment modéliser les connaissances et comment les partager ? Comment générer des plans d'actions ? Comment gérer des conflits ? Comment gérer la cohérence ? Comment permettre la communication et la négociation ?

Une nouvelle approche en IAD s'est développée durant les années 90 [Mae91, AR96]. Elle essaye de répondre à toutes ces questions et fait coopérer des entités auxquelles sont rattachées des caractéristiques de haut niveau. Ces entités seront désormais nommées *agents* et les systèmes correspondants sont appelés *systèmes multi-agents*.

1.3 Les agents

Les agents font partie d'un domaine de recherche relativement jeune : il n'existe pas encore de consensus sur la définition d'un agent. Cependant, la définition de Wooldridge et Jennings semble devenir celle de référence : « *Un agent est un système informatique situé dans un environnement, et capable d'actions autonomes afin d'atteindre ses objectifs de conception* » [WJ95].

Les deux propriétés principales d'un agent sont l'*autonomie* et le fait qu'il est *situé dans un environnement*. En IAD, un agent n'agit pas seul. Il existe un certain nombre d'autres agents dans son environnement, avec lesquels il interagit. L'*interactivité* est un troisième concept important.

1.3.1 L'autonomie des agents

L'autonomie signifie simplement qu'un agent est indépendant, qu'il est capable de prendre lui-même des décisions, sans l'intervention directe des humains,

[1] La littérature anglaise parle de *Distributed Artificial Intelligence (DAI)*.

ou d'autres agents. Il garde le contrôle de ses propres actions et de ses états internes. L'autonomie des agents implique que ce type de système a une tendance à être complètement décentralisé. L'autonomie présente deux aspects indépendants : l'autonomie dynamique et celle dite non-déterministe.

L'autonomie est dite *dynamique* parce que l'agent est capable d'exercer par lui-même un certain degré d'activité. Cette gradation du comportement peut varier de la passivité à une proactivité complète. Les agents peuvent réagir non seulement à des invocations de méthodes mais également à des évènements observables à l'intérieur de l'environnement. Les agents proactifs sont quant à eux capables de sonder l'environnement pour détecter des évènements ou d'autres messages afin de déterminer les actions qu'ils doivent prendre. En résumé, l'agent peut décider quand il agit ou non. Dans un système multi-agents, les agents peuvent être engagés dans des interactions multiples et parallèles avec d'autres agents, amplifiant encore plus l'aspect dynamique.

Les agents peuvent également présenter des comportements qui sont dans une certaine mesure imprévisibles — ou *non-déterministes*— . Le degré d'imprévisibilité peut varier de complètement prévisible à complètement imprévisible.

1.3.2 L'environnement d'un agent

La seconde propriété — situation dans un environnement — peut, à première vue, sembler moins spécifique au concept d'agent. En effet, tout logiciel peut être considéré comme étant situé dans un environnement. En réalité, c'est l'environnement qui est particulier. Il est dynamique, imprévisible, et incertain.

L'environnement est *dynamique*. Il change rapidement. L'agent ne peut pas le considérer comme étant statique pendant qu'il tente d'atteindre un objectif.

L'environnement est *imprévisible*. Il est dès lors impossible de prévoir son état futur. Cette impossibilité provient du fait que l'agent ne connaît pas parfaitement, ni complètement son environnement.

Enfin, l'environnement est *incertain*. Les actions que l'agent est capable de réaliser, peuvent éventuellement échouer pour des raisons que l'agent ne connaît pas. De plus, une même action peut avoir deux effets désirés différents à des moments différents dans un même environnement. On dit alors que l'environnement est non-déterministe.

1.3.3 L'interactivité des agents

L'interactivité d'un agent implique sa capacité à communiquer avec l'environnement et les autres entités. L'interaction peut se faire sous la forme d'une invocation. Un autre type d'interaction, plus complexe, inclut d'autres agents qui réagissent à des évènements de l'environnement et qui agissent sur cet environnement. Un agent s'engage dans des interactions multiples et parallèles avec les autres agents. Ce modèle de communication est généralement asynchrone.

Dès lors, un langage de communication inter-agents est nécessaire[2]. Il spécifie

[2] Aussi appelé *Agent Communication Language* (ACL) en anglais.

les discours, les ontologies et les formes[3] de conversation qu'on appelle protocole. Deux types d'ACL sont les plus populaires. Il s'agit de KQML[4] [FFMM94] et du FIPA ACL[5].

1.4 Les agents intelligents

Wooldridge fait la distinction entre un agent et un agent intelligent qui « *doit être davantage réactif, proactif et social* » [Woo02].

Un agent peut être réactif et/ou proactif. On dit qu'il est *réactif* si, en percevant les changements significatifs de son environnement, il y répond de manière opportune. Ce type d'agent réagit simplement en réponse à son environnement. Il est dirigé par les évènements[6]. « Un agent peut faire preuve de comportement opportuniste, être dirigé par les objectifs[7], et prendre des initiatives » [Woo02]. On dit alors que l'agent est *proactif*. Nous y reviendrons plus en détail dans la section 1.5.

Un agent intelligent est doué de *compétences sociales*. Notre monde réel est lui-même un environnement multi-agents. En tant qu'être humain, nous ne pouvons pas espérer atteindre nos objectifs sans tenir compte des autres. Certains de nos objectifs ne peuvent d'ailleurs être atteints qu'avec la coopération des autres. Il en va de même pour certains types d'environnements informatiques. La compétence sociale des agents est leur capacité à interagir et à coopérer avec les autres agents — et éventuellement avec des humains— via un langage de communication inter-agents (cfr. section 1.3.3). Ces interactions sont de haut niveau, et souvent vues en termes d'interactions humaines, comme des négociations, des coordinations, etc.

Un agent doit être *robuste* afin de pouvoir récupérer des échecs. Il est possible d'assurer cette robustesse en rendant l'agent plus flexible. Ce dernier doit connaître plusieurs façons d'atteindre un objectif donné.

Notons qu'un agent peut posséder un bon nombre d'autres propriétés secondaires. Par exemple, un agent doit posséder la *propriété de véracité*. Il ne peut pas communiquer d'informations fausses. Un agent peut être *rationnel*. Il agit dans un certain ordre pour atteindre ses objectifs et ne va pas agir de manière à empêcher que ses objectifs soient atteints. C'est pour répondre à cette caractéristique que le modèle *Belief-Desire-Intention* a été développé [Bra87]. Nous en reparlerons dans le chapitre 2, page 21. Il existe aussi des agents dits *mobiles*. Ces agents possèdent la capacité de pouvoir se déplacer sur un réseau électronique. Ils peuvent être démarrés sur un ordinateur et continuer leur tâche sur un autre ordinateur du même réseau si le premier est mis hors tension. Un *learning agent* peut apprendre par ses expériences afin d'améliorer ses performances.

Un agent intelligent est un composant logiciel possédant les caractéristiques suivantes [WP04]. Il est :

– *Situé* : il existe dans un environnement

[3] Appelées *patterns* en anglais.

[4] *Knowledge Query and Manipulation Language.*

[5] Spécifications disponibles sur le site `http://www.fipa.org/repository/aclspecs.html`.

[6] La littérature anglaise parle d'*event-directed agents.*

[7] Aussi dit *goal-directed.*

- *Autonome* : il est indépendant, et non contrôlé de l'extérieur
- *Réactif* : il répond aux changements de son environnement
- *Proactif* : il réalise des objectifs
- *Flexible* : il peut atteindre ses objectifs de différentes manières
- *Robuste* : il peut récupérer des échecs
- *Social* : il interagit avec d'autres agents

1.5 Agents réactifs et proactifs

Les *agents réactifs* n'ont pas d'intelligence propre. Ils n'ont pas de mémoire du passé. C'est la raison pour laquelle ils ne possèdent qu'un mécanisme simple de réaction aux évènements. Leurs décisions sont basées uniquement sur le présent. Ils ne réagissent qu'à des stimuli produits par l'environnement ou par d'autres agents et exécutent dès lors une fonction ou une procédure. L'agent en tant que tel n'est donc pas intelligent, c'est le système dans son ensemble qui peut être doué d'intelligence. Ce type d'agent est dirigé par les évènements. On dit de lui qu'il est « *event-directed* ».

Les *agents proactifs* sont quant à eux doués d'intelligence, en raison de leur capacité de raisonnement. Nous détaillerons cette capacité au chapitre 2, page 21. Les agents proactifs peuvent travailler de manière plus indépendante pour réaliser leurs tâches. Ils ont une mémoire du passé et peuvent planifier leur futur. Ils sont dirigés par les objectifs. Pour cette raison, ils sont dits « *goal-directed* ». Pour les atteindre, ils sont capables de prendre certaines initiatives, de coordonner leurs activités et de gérer des négociations. Ils connaissent différentes manières — appelées plans ou intentions— d'atteindre leurs objectifs et sélectionnent la plus appropriée en fonction de l'état de l'environnement à un moment donné. Ils sont capables de prévoir des réactions possibles de leurs actions. En cas d'échec d'un plan, ils peuvent toujours tenter d'atteindre un objectif d'une autre manière. Ce type d'agent peut également réagir à un stimulus de son environnement, mais il agit toujours en fonction de ses intentions propres.

1.6 Taxinomie des agents

Il est possible de classifier les agents en fonction des propriétés qu'ils possèdent. Tous les agents doivent au moins satisfaire les propriétés d'autonomie et de réaction à l'environnement. Ensuite, en ajoutant d'autres propriétés, nous obtiendrons d'autres types d'agents. Par exemple, un agent proactif et capable de comportement social sera un agent intelligent. Il existe toute une série de propriétés que nous avons déjà citées auparavant. Ces propriétés sont assez indépendantes les unes des autres. Par exemple, un agent peut être soit mobile, soit apprenant, soit les deux. Les agents peuvent dès lors être classés sous forme d'ensembles en fonction de leurs propriétés.

Une autre classification possible peut être réalisée en fonction des tâches que les agents sont capables de réaliser.

Le modèle de classification le plus original est sans nul doute celui de Franklin et Graesser [FG96]. Leur modèle de classification des agents est assez similaire au modèle de classification biologique. Le modèle biologique prend la forme d'un arbre des créatures vivantes à la racine avec des espèces aux feuilles. Celui de Franklin et Graesser, prend la forme d'un arbre avec les agents autonomes à la racine. Au premier niveau, les agents autonomes sont séparés en agents biologiques, robotiques ou informatiques. Au deuxième niveau, les agents informatiques sont soit logiciels soit de vie artificielle. Enfin, au troisième niveau les agents logiciels sont soit des agents dédiés à une tâche, soit des agents de divertissement, soit des virus. Franklin et Graesser mentionnent que leur classification peut être éventuellement étendue.

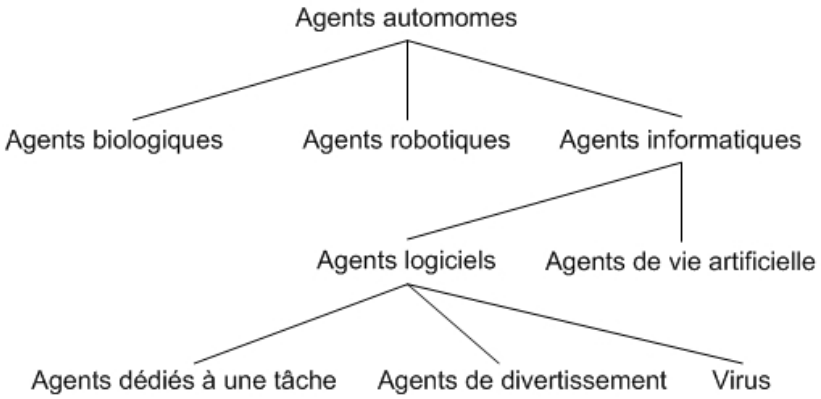


FIG. 1.1 – Taxonomie des agents selon Franklin et Graesser [FG96]

1.7 Agents et Objets

Il est souvent tentant de vouloir définir les agents comme des objets. J. Bradshaw définit un agent comme un objet avec des attitudes [Bra97]. L'agent peut dès lors avoir des comportements, une certaine mobilité, communiquer, etc. Certains auteurs considèrent même que les agents ne sont que des objets. D'autres au contraire, voient les agents et les objets de manière très différente malgré un certain nombre de propriétés communes. Les deux approches envisagent cependant les agents et objets comme deux notions qui impliquent une manière différente de penser le développement de logiciels.

La figure 1.2 illustre une manière de voir l'évolution des différentes approches de la programmation que l'on retrouve souvent dans la littérature.

Au début de la programmation, l'unité de base était le programme complet. Il n'existait pas d'unité réutilisable et le concept de modularité n'était pas encore d'application. L'état d'un programme était de la responsabilité de son programmeur et l'invocation du programme était faite par le système d'exploitation.

Les programmes devinrent de plus en plus complexes et les programmeurs introduisirent le concept de modularité afin de mieux organiser leur code. Il existait dès lors de petites unités de code — appelées aussi procédures — qui pouvaient

	Programmation monolithique	*Programmation modulaire*	*Programmation par objets*	*Programmation par agents*
Comportement	non modulaire	modulaire	modulaire	modulaire
Etat	externe	externe	interne	interne
Invocation (et choix)	externe	externe (appel)	externe (message)	Interne (règles, buts)

FIG. 1.2 – Evolution des différentes approches de la programmation [PD97]

être réutilisées dans de nombreuses situations. L'état des procédures était déterminé par l'intermédiaire d'arguments extérieurs et l'invocation se faisait par une instruction d'appel.

La programmation orientée objets a ajouté à l'approche modulaire le concept d'encapsulation. Les segments de codes, maintenant appelés méthodes, permettent de garder un contrôle local sur ses variables qui ne sont manipulées que par ses méthodes. Un objet encapsule un état, communique, et possède des méthodes correspondant aux opérations qui peuvent être exécutées sur cet état. Les méthodes d'un objet sont invoquées par des objets extérieurs— ou parfois par l'objet lui-même — via l'envoi d'un message.

Le paradigme orienté agents poursuit les mêmes objectifs que le paradigme orienté objets. Dans les deux cas, un développement fiable est facilité par l'encapsulation d'un état dans des modules qui contiennent les définitions et les structures de données requises pour fonctionner de manière indépendante. Les agents étendent le concept d'encapsulation, et possèdent une représentation des comportements d'un plus haut niveau que l'approche orientée objets. Ils possèdent leur propre *thread*[8] de contrôle et l'invocation est réalisée par eux-mêmes. On dit alors qu'ils sont doués d'autonomie, ou encore d'initiative. Wooldridge et Jennings font une analogie entre l'autonomie d'un agent et l'encapsulation d'un objet. « *Un objet encapsule des états et garde le contrôle de cet état. Celui-ci ne peut être modifié que via des méthodes que l'objet fournit. De la même manière, un agent encapsule un état. Nous pensons aussi qu'un agent encapsule un comportement en plus d'un état. Un objet quand à lui n'encapsule pas de comportement. Il n'a pas le contrôle de l'exécution de ses méthodes. Dans ce sens, un objet n'est pas autonome* » [JW98].

Les agents différent des objets principalement pour trois raisons.

Premièrement, les agents sont autonomes. Ils expriment plus fortement la no-

[8] Processus léger exécuté sur une machine.

tion d'autonomie que les objets. En particulier, ils décident eux-mêmes s'ils exécutent ou non une action à la demande d'un autre agent. Les objets sont par nature passifs. En effet, le terme autonomie ne s'applique pas à une entité dont l'invocation dépend uniquement d'autres composants du système. Cependant, UML et Java ont introduit des mécanismes tels que des *event-listeners* de manière à permettre aux objets d'être un peu plus réactifs. Alors que les agents sont non-déterministes, l'usage et le support qu'offrent les langages orientés objets reste dans une approche plus prévisible, de manière à faciliter la réutilisabilité des composants. En effet, quand un message spécifique est envoyé à un objet, la méthode qui sera invoquée est entièrement prévisible. L'objet pourrait éventuellement décider de ne pas traiter le message, mais dans ce cas on considère qu'il s'agit d'une erreur. Avec les agents, ce n'est pas le cas.

Deuxièmement, les agents sont intelligents. Ils sont doués d'un comportement flexible (réactivité, proactivité, sociabilité), alors que le paradigme orienté objets ne permet pas ce genre de comportement.

Et troisièmement, les agents sont actifs. Un système multi-agents est intrinsèquement *multi-thread*, dans le sens où chaque agent est supposé posséder au moins un *thread* de contrôle actif. Il n'existe pas de *thread* central qui ordonne tous les autres puisque les agents s'organisent eux-mêmes [Ode02].

En plus de ces différences principales, il existe une série d'autres différences liées aux caractéristiques secondaires des agents. Ces caractéristiques ne sont pas toujours appliquées à l'orienté agents, et le sont parfois à l'orienté objets.

Les applications objets sont généralement centralisées. Certaines situations nécessitent pourtant un haut degré de décentralisation. Il existe des techniques objets pour mettre en oeuvre la décentralisation. Mais au contraire des agents, les objets ne sont pas conçus pour cela.

Dans les langages orientés objets, les objets sont créés à partir d'une classe. Une fois créés, ils ne peuvent jamais changer de classe ou devenir l'instance de plusieurs classes (à l'exception des classes dont ils héritent). Les agents fournissent une approche plus flexible. En effet, ils possèdent des capacités. Ces capacités leur permettent de jouer un rôle précis puis de changer. Les agents peuvent jouer des rôles variés dans différents domaines. Cette approche multiple et dynamique, permettant plus de flexibilité, est plus proche de notre modèle de perception du monde. Alors que les caractéristiques d'un objet sont définies dans sa classe, un agent peut acquérir ou modifier ses caractéristiques. Autrement dit, les agents ont la capacité d'apprendre. Ils peuvent adapter leurs comportements et agir ainsi différemment des autres agents.

L'autonomie et le caractère interactif des agents ressemblent plus aux systèmes naturels que les objets. Ils s'organisent en société, en hiérarchie, etc. La nature peut fournir un grand nombre d'idées pour la conception de systèmes multi-agents.

1.8 Utilité des agents

Nous avons déjà dit que les agents sont autonomes. Ceci implique l'encapsulation des invocations. Lorsqu'un message est envoyé à un agent, celui-ci — qui est

autonome — garde le contrôle du traitement du message. Ceci signifie que l'agent se charge d'un objectif, et qu'il garde la responsabilité de l'objectif jusqu'à ce que celui-ci soit atteint. L'agent peut être vu comme un employé qui a le sens des responsabilités et prend des initiatives. La supervision de ce type d'employé exige une communication et un couplage réduit entre les différents modules du système. Le couplage est généralement réduit pour un système plus décentralisé tel qu'un système orienté agents.

Les agents sont appropriés dans des situations où l'environnement est dynamique, imprévisible et incertain, et dans lequel les agents doivent pouvoir traiter les échecs possibles. Les systèmes multi-agents proactifs et réactifs sont plus « *human-like* » dans la manière dont ils résolvent les problèmes. Ce type de systèmes est utile dans les domaines où le logiciel doit se substituer aux humains. « *Les agents sont bien appropriés au développement d'applications distribuées complexes* » [Jen01]. En effet, ils fournissent l'abstraction et la décomposition nécessaire pour ce type de systèmes complexes. Ils proposent une meilleure interopérabilité, plus de flexibilité, ainsi qu'une coordination plus évoluée que les architectures logicielles classiques.

1.9 Concepts liés aux agents

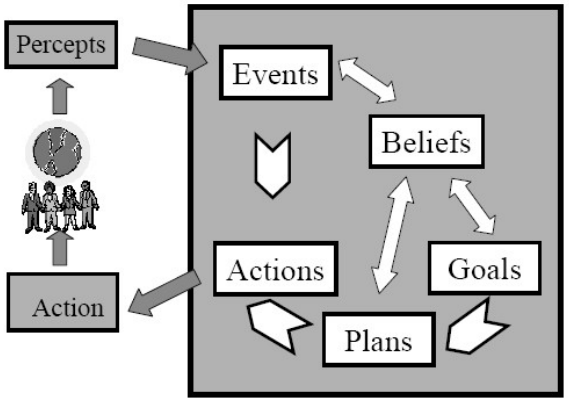


FIG. 1.3 – Les concepts liés aux agents [WP04]

1.9.1 Concepts liés à la situation d'un agent dans un environnement

Rappelons tout d'abord que l'une des principales propriétés d'un agent est sa situation dans un environnement. Les deux concepts définis pour expliquer les interactions entre un agent et son environnement sont les perceptions et les actions. La figure 1.3 nous montre les concepts liés aux agents.

Une *perception* est une information reçue de l'environnement par un capteur. Une *action* est ce que l'agent réalise. De par sa situation dans un environnement,

les actions que l'agent réalise affectent cet environnement.

1.9.2 Concepts liés à la réactivité et à la proactivité d'un agent

Un agent réactif répond simplement à un changement dans son environnement, tandis qu'un agent proactif est dirigé par des objectifs. Il est donc nécessaire de définir la notion d'objectif (*goal*), et d'événement (*event*).

Un *objectif* est ce sur quoi l'agent travaille. Souvent, les objectifs sont définis comme des états du monde auxquels l'agent veut arriver. Cependant, cette définition ne tient pas compte de certains types d'objectifs comme les contraintes de sécurité. Ces objectifs donnent à l'agent son autonomie et son caractère proactif. Les objectifs d'un agent sont dits persistants car si un plan qu'il exécute échoue, l'agent va considérer d'autres alternatives pour atteindre les objectifs en question.

Un *évènement* est une occurrence signifiante à laquelle un agent doit répondre. Bien que les événements sont souvent extraits des perceptions, certains peuvent être générés par l'agent lui-même. Un évènement déclenche un nouvel objectif, peut influencer l'environnement, et peut déclencher des actions qui seront réalisées immédiatement. Les perceptions peuvent être vues comme un type particulier d'événements qui sont générés par l'environnement. Elles peuvent être interprétées comme des données brutes.

1.9.3 Concepts pour répondre aux changements rapides de l'environnement

Les agents sont incapables de saisir entièrement tout l'environnement qui les entoure. Pour cette raison, l'agent doit maintenir en mémoire un certain nombre d'informations qu'il reçoit. C'est ce qu'on appelle les croyances (*beliefs*). Une *croyance* est une connaissance de l'agent sur son environnement, sur lui-même ou sur d'autres agents.

Pour répondre à ses objectifs, l'agent possède des plans. Un *plan* est une manière de réaliser un objectif. Il est écrit par le développeur et spécifie une série d'étapes à suivre. Un plan pour résoudre un objectif fournit une fonction qui indique si le plan est applicable ou non à une situation courante, ainsi que le corps du plan qui est exécuté. Un plan peut décomposer un objectif en sous-objectifs.

1.9.4 Concepts liés à l'aspect social des agents

Nous avons déjà souligné l'importance de l'aspect social des agents et des communications inter-agents. Il existe différentes formes d'interactions allant du message simple à un protocole bien défini.

Un *message* est le concept de base pour la communication inter-agents. Un message se définit comme une communication simple (*one-way*).

Le concept de *protocole* se définit quant à lui comme un pattern d'interaction (une séquence acceptable de messages qui forme une conversation).

1.9.5 Cycle d'exécution

Les concepts d'action, de perception, d'événement, d'objectif, et de croyance sont liés entre eux par le cycle d'exécution qui implémente le processus de prise de décision de l'agent.

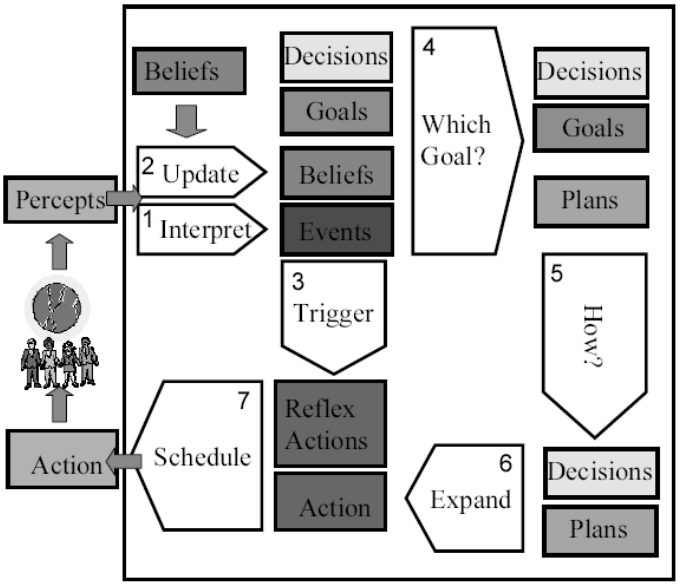


FIG. 1.4 – Le cycle d'exécution d'un agent [HPW01]

La figure 1.4 nous montre le cycle d'exécution qui décrit comment les concepts précédents interagissent quand un agent s'exécute. Il est composé des étapes suivantes [WP04] :

1. Les perceptions sont interprétées (selon les croyances) afin d'en extraire les évènements.
2. Les croyances sont mises à jour avec les nouvelles informations extraites des perceptions.
3. Les évènements produisent des actions et/ou de nouveaux objectifs.
4. Les objectifs sont mis à jour.
5. S'il n'y a pas de plan sélectionné pour l'objectif, ou si le plan échoue, ou encore s'il faut le reconsidérer, alors un (autre) plan est choisi.
6. Le plan choisi est étendu afin de pouvoir produire des actions.
7. Les actions sont planifiées et exécutées.

Les actions exécutées agissent sur l'environnement, provocant ainsi de nouveaux évènements, objectifs, croyances ou actions.

1.10 Architecture abstraite des agents

Il est possible de formaliser tous les concepts liés aux agents à un niveau plus abstrait [Woo02]. Cette vue abstraite des agents est appelée l'architecture abstraite des agents. L'*environnement* peut être défini comme un ensemble d'états instantanés et discrets :

$$E = \{e, e', ...\}$$

Cette hypothèse peut être faite, car même si l'environnement n'est pas réellement discret, il est possible de le modéliser de manière discrète avec le degré de précision souhaité.

Soit Ac l'ensemble fini des actions que l'agent peut exécuter. C'est ensemble est parfois appelé *compétence* de l'agent :

$$Ac = \{\alpha, \alpha', ...\}$$

On suppose que l'environnement est dans un certain état au démarrage de l'agent. A ce moment, l'agent choisit une action à effectuer dans l'état initial. Une fois l'action effectuée l'environnement se trouve dans un nouvel état. Sur base de cet état l'agent peut effectuer une nouvelle action, et ainsi de suite. La *trace* r (run) d'un agent dans un environnement est donc une séquence finie d'états de l'environnement et d'actions :

$$r = e_0 \overset{\alpha_0}{\rightarrow} e_1 \overset{\alpha_1}{\rightarrow} e_2 \overset{\alpha_2}{\rightarrow} ... \overset{\alpha_{u-1}}{\rightarrow} e_u$$

Soit R l'ensemble de toutes les séquences finies possibles sur l'environnement E et la compétence Ac ; R^{Ac} est alors le sous-ensemble des séquences qui se terminent par une action et R^E est le sous-ensemble de celles qui se terminent par un état.

Dans le but de modéliser l'effet d'une action sur l'environnement, nous définissons une fonction transformatrice d'état qui fait correspondre une trace se terminant par une action avec un ensemble possible d'états de l'environnement.

$$\tau : R^{Ac} \rightarrow \wp(E)$$

De cette manière, l'environnement est présumé dépendant de l'historique de l'agent. L'état prochain de l'environnement n'est en effet pas uniquement déterminé par l'action exécutée par l'agent et par l'état courant. Les actions réalisées par l'agent auparavant affectent aussi la détermination de l'état suivant. Cette définition montre en quoi l'environnement de l'agent peut être non-déterministe. Le résultat d'une action sur un état n'est donc pas entièrement prévisible.

Si $\tau(r) = \emptyset$ pour tout $r \in R^{Ac}$, alors il n'y a pas d'état successeur possible pour r et on dit que le système termine sa trace. On peut admettre que toutes les traces se terminent à un moment ou un autre.

Un environnement Env est un triplet tel que $Env = \langle E, e_0, \tau \rangle$, ou E est un ensemble des états possibles de l'environnement , $e_0 \in E$ est l'état initial, et τ est la fonction transformatrice d'état.

Les agents peuvent être modélisés par une fonction qui associe une action à une trace se terminant par un état :

$$Ag : R^E \rightarrow Ac$$

Un système agent est une paire composée d'un agent et d'un environnement. L'ensemble des différentes traces d'un agent Ag dans un environnement Env est noté $R(Ag, Env)$.

La séquence :

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, ...)$$

représente une trace de l'agent Ag dans l'environnement $Env = \langle E, e_0, \tau \rangle$ si :
- e_0 est l'état initial de Env
- $\alpha_0 = Ag(e_0)$
- pour tout $u > 0$,

$$e_u \in \tau(e_0, \alpha_0, ..., e_{u-1}, \alpha_{u-1})$$

où

$$\alpha_u = Ag(e_0, \alpha_0, ..., \alpha_{u-1}, e_u)$$

Deux agents Ag_1 et Ag_2 sont comportementalement équivalents dans un environnement Env, ssi $R(Ag_1, Env) = R(Ag_2, Env)$.

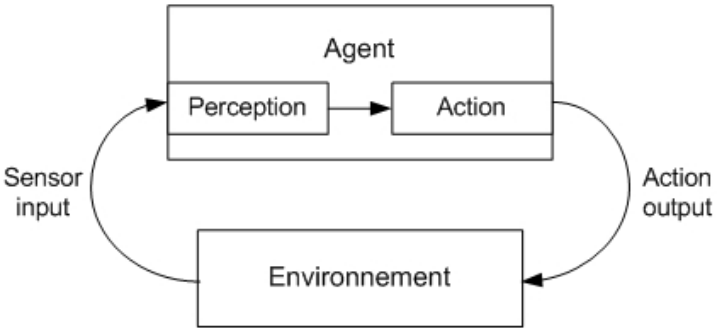


FIG. 1.5 – Vue abstraite d'un agent réactif

Un agent réactif (figure 1.5) prend des décisions entièrement basées sur le présent. Son comportement peut être défini par la fonction :

$$Ag : E \rightarrow Ac$$

Nous devons encore compléter la vue abstraite d'un agent par les définitions des termes perception et action.

La perception est une fonction qui représente la capacité de l'agent à observer son environnement. La fonction de perception est une fonction d'un état de l'environnement vers une perception. Les agents situés dans le monde réel peuvent

percevoir en utilisant des caméras, des micros, etc. Soit Per un ensemble non vide de perceptions. La fonction de perception est alors :

$$perception : E \rightarrow Per$$

L'action est une fonction qui représente le processus de prise de décision de l'agent. Cette fonction fait correspondre une séquence de perceptions à une action :

$$action : Per^* \rightarrow Ac$$

Un agent peut désormais être défini comme une paire $Ag = \langle perception, action \rangle$. Grâce à cette nouvelle définition, nous pouvons identifier deux propriétés importantes de la perception des agents.

1. Soit deux états différents e_1 et e_2 du même environnement $e_1, e_2 \in E$ tel que $perception(e_1) = perception(e_2)$. Nous avons dès lors deux états différents de l'environnement qui sont associés à la même perception. e_1 et e_2 sont indiscernables du point de vue de l'agent.
2. Définissons $\sim$ la relation d'équivalence sur E. Soit $e_1, e_2 \in E$, Soit $e_1 \sim e_2$ est équivalent à $perception(e_1) = perception(e_2)$. L'environnement est accessible pour l'agent ssi $|E| = |\sim|$ et l'agent est dit omniscient. Si $|\sim| = 1$, l'agent n'a pas de capacité de perception. L'agent perçoit l'environnement comme étant monovalent. Il ne perçoit les états de l'environnement que d'une seule manière, parce que tous les états sont identiques de son point de vue perceptif.

Les agents proactifs (figure 1.6) ont une sorte de structure de données internes pour enregistrer des informations sur des états de l'environnement ainsi qu'un historique. Soit I l'ensemble de tous les états internes possibles d'un agent. Le processus de décision doit maintenant tenir compte de cette nouvelle information. Nous devons dorénavant redéfinir la fonction d'action qui associe une action à un état interne :

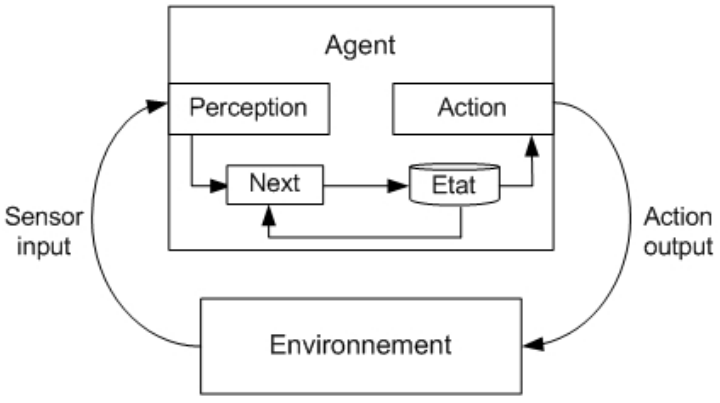


FIG. 1.6 – Vue abstraite d'un agent proactif

$$action : I \rightarrow Ac$$

Une nouvelle fonction *next* doit être introduite. Elle associe à un état interne et une perception, un nouvel état interne :

$$next : I \times Per \rightarrow I$$

Un agent proactif démarre d'un état interne $i_0 \in I$, il observe alors un état e_0 de l'environnement qui génère une perception $perception(e_0)$. L'état interne est mis à jour, $i_1 = next(i_0, perception(e_0))$. L'agent choisit ensuite l'action la plus appropriée $action(i_1)$. Il agit sur l'environnement qui passe dans l'état e_1. L'agent reçoit alors une nouvelle perception et réagit avec le même mécanisme.

1.11 Les limites des solutions agents

La pléthore de définitions de la notion d'agent montre à quel point celle-ci est encore récente et sans véritable consensus. L'IAD doit encore dépasser un certain nombre de limitations. Elle doit tout d'abord donner une définition unanime de cette notion. Elle doit encore se donner un ensemble cohérent de principes pour la structuration des agents et consolider ces différentes techniques et méthodologies de conception.

Enfin, certains pourraient penser que les agents n'apportent rien de nouveau à l'informatique et prétendre que les agents ne sont que des objets, qu'un système développé avec des agents aurait probablement pu être développé avec des technologies plus conventionnelles. Au contraire, nous pensons que les agents offrent une nouvelle approche de plus haut niveau pour la conception de systèmes informatiques. L'intérêt de cette approche est la simplification de conception qu'elle apporte pour certaines classes de problèmes.

Chapitre 2

Modèle BDI

Sommaire

2.1 Introduction

Le modèle *Belief-Desire-Intention* (BDI) est un des modèles les plus célèbres permettant de mettre en pratique le raisonnement d'agents. A l'origine, il fut développé par Michael Bratman [Bra87] puis formalisé par Rao et Georgeff[1] [RG91]. L'apparition de ce modèle a fait évoluer considérablement la recherche sur les agents et sur l'intelligence artificielle en général.

Son succès est principalement dû au fait qu'il est assez proche du modèle du raisonnement humain. En effet, l'architecture BDI se base sur une vue anthropomorphique du concept d'agent, en lui attribuant des attitudes mentales telles que des croyances, des désirs et des intentions, représentant respectivement l'information, la motivation et l'état délibératif de l'agent. Ces attitudes mentales

[1] La formalisation du modèle BDI est présentée dans l'annexe A

déterminent le comportement du système et sont critiques pour que l'agent atteigne de manière adéquate et optimale son objectif.

Ce modèle de raisonnement est particulièrement bien adapté au développement d'agents proactifs. La plate-forme Jack que nous avons utilisée lors de notre stage à l'UTS est basée sur ce modèle de raisonnement.

2.2 Vue globale de l'architecture BDI

Un agent possède des *croyances* sur le monde, des *désirs* à satisfaire et agit avec des *intentions*. Ces croyances, désirs, et intentions sont appelés attitudes mentales — ou état mental— de l'agent [GR98]. L'architecture de tout système BDI est basée sur des structures de données comprenant les trois attitudes mentales et une librairie de plans. Une *file d'évènements* est utilisée pour stocker temporairement les évènements perçus par l'agent.

2.2.1 Les croyances (*Beliefs*)

Les *croyances* (beliefs) représentent les connaissances du monde. « Cependant, d'un point de vue informatique, elles ne sont qu'une manière de représenter un état du monde, pouvant être une valeur d'une variable, une base de données ou une expression logique. » [GPP+99] Ces croyances sont indispensables parce que l'environnement est dynamique, et parce que le système ne peut avoir qu'une vue partielle du monde. Deux activités de l'agent peuvent mettre à jour les croyances. Il s'agit des perceptions de l'environnement et de l'exécution des intentions de l'agent.

2.2.2 Les désirs (*Desires*)

Les *désirs* (desires) — aussi appelés *objectifs* (goals)— sont un autre élément essentiel du système. Ils correspondent aux tâches qui sont affectées à l'agent. A chaque instant, le système peut potentiellement accomplir un certain nombre d'objectifs différents. De nouveau, « d'un point de vue informatique, ils ne représentent qu'une valeur de variable ou une expression logique » [GPP+99].

2.2.3 La file d'événements

Les perceptions de l'agent qui correspondent à des *évènements* sont stockées temporairement dans une file. Ces évènements sont de trois types : l'acquisition ou la suppression de croyances, la réception de message et l'acquisition d'un nouvel objectif.

2.2.4 Les plans (*Plans*)

Les agents BDI possèdent généralement une librairie de *plans* prédéfinis. Ces plans précisent, sous la forme d'une formule mathématique, les circonstances dans lesquelles ils peuvent être appliqués. Le corps du plan représente les parcours possibles des actions. Il peut être représenté sous la forme d'un arbre où chaque noeud

est considéré comme un état et où chaque branche représente les actions ou les sous-objectifs de l'agent. Les sous-objectifs sont postés dans la file d'évènements lors de l'exécution du plan, ensuite d'autres plans seront considérés lors du traitement de ces évènements. Un plan peut aussi être une simple séquence linéaire d'actions. Les plans qui atteignent le mieux les différents objectifs sont dépendants de l'état de l'environnement mais sont indépendants de l'état interne du système.

Plus simplement, les plans sont des procédures précompilées possédant un ensemble de conditions qui déterminent leurs cas d'applications.

2.2.5 Les intentions (*Intentions*)

Les *intentions* sont les parcours courants d'actions que l'agent a décidé d'exécuter. Chaque intention est implémentée sous la forme d'une pile d'instance de plans. Elle représente l'engagement d'un agent à exécuter un plan. Lorsqu'une instance de plan est créée, il y a deux possibilités :

- soit le plan est instancié à la suite d'un évènement extérieur. Dans ce cas, une pile vide est créée pour cette intention de l'agent ;
- soit le plan est instancié à la suite d'un évènement interne, ce qui signifie que c'est une intention qui a généré cet évènement. Dans ce cas, l'instance du plan est placée sur la pile correspondant à l'intention qui a généré l'évènement.

« D'un point de vue informatique, les intentions peuvent être un ensemble de threads d'un processus qui sont susceptibles d'être interrompus jusqu'au moment où ils reçoivent une information significative de leur environnement » [GPP+99].

2.2.6 L'interpréteur BDI

Un interpréteur agit sur ces structures de données. Ces composants sont représentés sur la figure 2.1. Différents interpréteurs peuvent utiliser différents algorithmes en fonction de la stratégie souhaitée. Voici un exemple d'algorithme simple [Her03] :

1. Mettre à jour la file d'évènements par la perception et les actions internes ;
2. Sélectionner le premier évènement de la file et générer les nouveaux désirs en cherchant un ensemble de plans appropriés dans la librairie ;
3. Sélectionner dans l'ensemble de plans celui qui correspond le mieux aux croyances de l'agent, et créer une instance de ce plan ;
4. Empiler l'instance du plan dans une nouvelle pile ou dans une pile existante suivant que le plan est instancié suite à un évènement extérieur ou interne ;
5. Exécuter l'étape courante de la pile d'intentions courantes — s'il s'agit d'une action, il suffit de l'exécuter, sinon il s'agit d'un sous-objectif et il faut le poster dans la file d'évènement — .

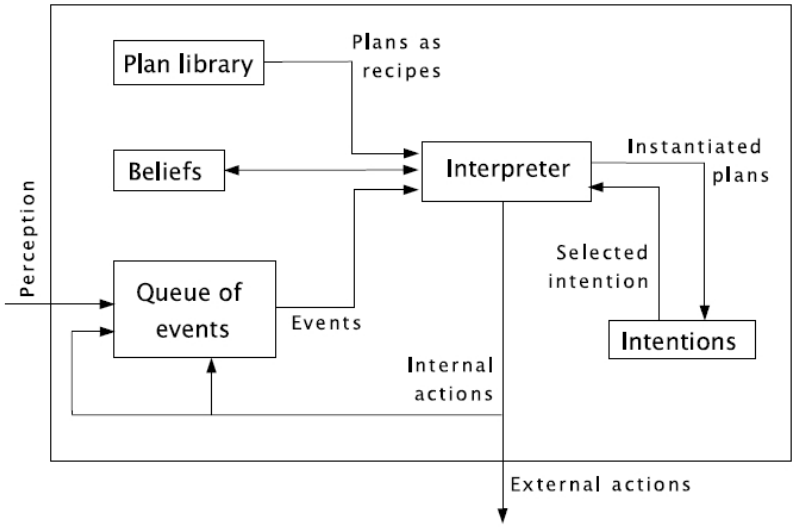


FIG. 2.1 – Vue globale d'un agent BDI [Her03]

2.2.7 Le choix du plan à exécuter

Dans cette section, nous présentons un algorithme qui choisit le plan le plus approprié pour atteindre un objectif fixé. Les agents Jack — détaillés à la section 2.4 page 29 — choisissent les plans selon un algorithme similaire à celui-ci :

1. Déterminer les plans qui sont appropriés dans une librairie de plans ;
2. Déterminer le sous-ensemble des plans appropriés qui sont applicables ;
3. Sélectionner un des plans applicables ;
4. Exécuter ce plan.

Un plan est dit *approprié*[2] s'il spécifie qu'il est capable d'atteindre l'objectif poursuivi. Un plan approprié est dit *applicable*[3] s'il y a un sens de l'appliquer à la situation présente. L'applicabilité est spécifiée dans une condition tenant compte des croyances de l'agent. Cette condition est appelée le *contexte* du plan. En pratique, vérifier si un plan est applicable à une situation consiste à vérifier si sa *condition de contexte* est vraie pour cette situation. Cette condition doit être implémentée par le programmeur.

L'exécution d'un plan peut réussir. Dans ce cas, l'objectif ainsi que tous ces sous-objectifs ont été atteints. L'exécution d'un plan peut aussi échouer. Dans ce cas, l'agent va essayer d'atteindre l'objectif en reconsidérant d'autres alternatives, d'autres plans. L'agent doit alors réévaluer l'applicabilité des plans, sélectionner un plan alternatif et l'exécuter.

Si tous les plans applicables échouent, alors l'objectif échoue également. Si un sous-objectif échoue, l'échec est propagé à l'objectif père.

[2] *Relevant* en anglais.
[3] *Applicable* en anglais.

Le choix de plan pour atteindre un objectif ou un sous-objectif prend en compte l'état courant de l'environnement. L'agent est d'autant plus *flexible* que le nombre de plans qui peuvent être utilisés pour atteindre un (sous-)objectif est élevé. Si un plan échoue, l'agent peut essayer des alternatives. On dit qu'il est *robuste*.

Différentes façons d'atteindre un objectif

Chaque objectif possède un certain nombre de plans permettant de l'atteindre. Chaque plan peut être composé de plusieurs sous-objectifs pour lesquels il existe un certain nombre de plans applicables. Nous pouvons représenter cette hiérarchie sous forme d'un arbre objectif-plan comme sur la figure 2.2. Les enfants d'un objectif sont des alternatives pour l'atteindre tandis que les enfants d'un plan sont des sous-objectifs. Un objectif peut donc être associé à un noeud OR et un plan est associé à un noeud AND.

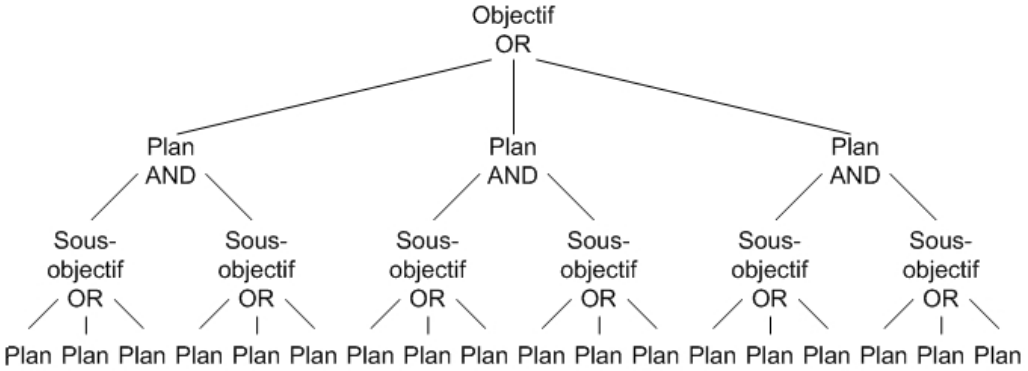


FIG. 2.2 – Arbre objectif-plan de profondeur 2

Si nous notons C le nombre de plans applicables pour chaque objectif, par S le nombre de sous-objectifs pour chaque plan et par D la profondeur de l'arbre, alors le nombre de manières par lesquelles l'objectif de la racine peut être atteint est

$$C^{((S^D-1)/(S-1))} \tag{2.1}$$

Sauf si $S = 1$ et dans ce cas, il s'agit de C^D. Cette formule est démontrée à la section suivante.

Par exemple, si $C = 2$, $S = 4$ et $D = 3$,
$C^{((S^D-1)/(S-1))} = 2^{((4^3-1)/(4-1))} = 2^{(64-1)/3} = 2^{21} = 2\,097\,152$.

Comme l'illustre la figure2.2, cette manière de programmer permet un très grand nombre de possibilités d'atteindre un objectif.

2.2.8 Calcul du nombre de possibilités

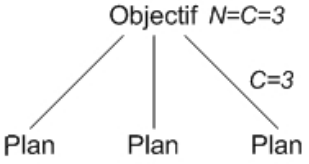
Nous voulons déterminer le nombre de manières différentes d'atteindre un objectif. Pour simplifier, nous considérerons que l'arbre objectif/plan est uniforme,

dans le sens où un objectif possédera toujours le même nombre de plans applicables et où chaque plan sera toujours composé du même nombre de sous-objectifs, à l'exception des plans des feuilles.

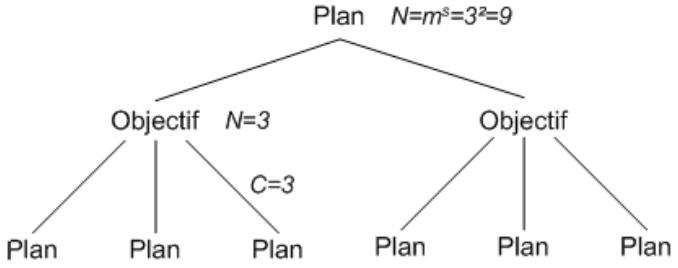
Soit :

- C le nombre de plans applicables pour chaque objectif ;
- S le nombre de sous-objectifs par plan ;
- D la profondeur de l'arbre.

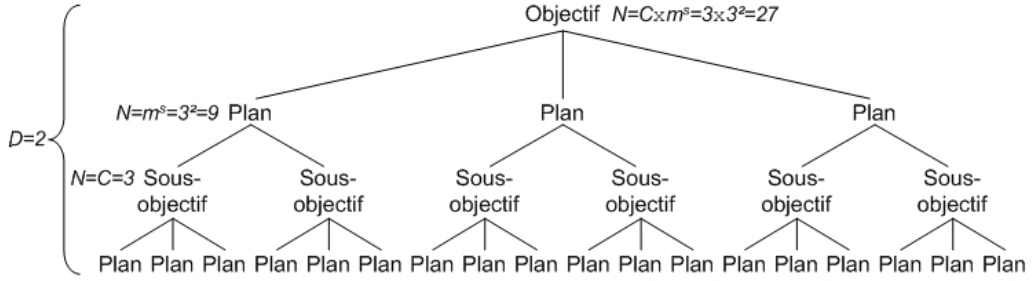
Soit un objectif G qui peut être atteint par C plans applicables. Les plans ne possèdent pas de sous-objectifs. G peut être atteint de C manières.



Soit P un plan composé de S sous-objectifs qui eux-mêmes peuvent être atteints de m manières possibles. Chaque sous-objectif doit être atteint séparément. Le nombre de manières d'exécuter P est le produit $m \times \cdots \times m$, c'est-à-dire m^S.



Soit G un objectif qui peut être atteint par C plans applicables, pouvant chacun être exécuté de p manières différentes. Chaque plan est une alternative et le nombre de manières d'atteindre G est la somme $p + \cdots + p$, c'est-à-dire $C \times p$.



Nous formalisons ceci en définissant $\Delta_G(D)$ qui est le nombre de manières d'atteindre un objectif à la racine d'un arbre de profondeur D, et $\Delta_P(D)$ qui est le nombre de manières d'exécuter un plan à la racine d'un arbre de profondeur D.

Nous avons alors :

$$\Delta_G(1) \quad = \quad C$$

$$\begin{aligned}\Delta_P(D+1) &= \Delta_G(D)^S \\ \Delta_G(D+1) &= C \times \Delta_P(D+1) \\ &= C \times \Delta_G(D)^S\end{aligned}$$

En dépliant cette définition, nous obtenons :

$$\begin{aligned}\Delta_G(1) &= C \\ \Delta_G(2) &= C \times \Delta_G(1)^S \\ &= C \times C^S \\ &= C^{S+1} \\ \Delta_G(3) &= C \times \Delta_G(2)^S \\ &= C \times C^{(S+1)^S} \\ &= C \times C^{(S+1)\times S} \\ &= C \times C^{S^2+S} \\ &= C^{S^2+S+1}\end{aligned}$$

De manière plus générale, pour un arbre de profondeur D, nous aurons :

$$\Delta_G(D) = C^{S^{D-1}+\cdots+S^2+S+1}$$

La somme $\sum = C^{S^{D-1}+\cdots+S^2+S+1}$ peut être simplifiée.

$$\begin{aligned}\sum &= S^{D-1} + \cdots + S^2 + S + 1 \\ S \times \sum &= S^D + \cdots + S^2 + S \\ (S \times \sum) - \sum &= S^D - 1 \\ (S-1) \times \sum &= S^D - 1 \\ \sum &= \frac{S^D - 1}{S - 1}\end{aligned}$$

(Si $S = 1$ alors $\sum = 1 + 1 + \ldots + 1 = D$)

$\Delta_G(D) = C^{S^{D-1}+\cdots+S^2+S+1}$ devient alors $C^{((S^D-1)/(S-1))}$ (sauf si $S = 1$, et dans ce cas $\Delta_G(D) = C^D$), et nous retrouvons la formule 2.1.

2.3 De l'arbre de décisions aux mondes possibles

Les agents proactifs sont caractérisés par un environnement non déterministe, dans le sens où ce dernier peut évoluer de différentes manières. De plus, ils sont eux-mêmes non-déterministes car ils peuvent exécuter différents plans et actions pour un même état de l'environnement à deux moments différents.

Pour agir, il faut sélectionner un plan approprié pour atteindre l'objectif. Une telle fonction de sélection a besoin d'au moins deux types d'*inputs*. Premièrement, il est nécessaire de disposer des informations sur l'état de l'environnement (les *beliefs*). Deuxièmement, nous avons besoin d'informations sur les objectifs à accomplir (les *desires*).

2.3.1 Arbre de décisions

Nous pouvons modéliser le comportement des agents proactifs sous la forme d'une structure d'arbre dans lequel chaque branche représente une alternative d'exécution. Chaque noeud représente un certain état du monde et chaque transition correspond à une action réalisée par le système ou à un évènement produits par l'environnement. Nous différencions les noeuds de décision des noeuds de possibilité, tout comme nous différencions les actions prises par le système des évènements produits par l'environnement.

De manière informelle, un arbre de décisions est composé de noeuds de décision, de noeuds de possibilité et de noeuds terminaux, et inclut une fonction de probabilité qui fait correspondre des probabilités à un noeud de possibilités, ainsi qu'une fonction *payoff* qui fait correspondre des noeuds terminaux à des nombres réels. Enfin, une fonction de délibération, du type maximisation de l'utilité, est définie pour choisir une ou plusieurs actions à réaliser pour un noeud donné.

Le modèle des mondes possibles considère que les croyances d'un agent peuvent être vues comme un ensemble de mondes possibles, pouvant eux-mêmes être représentés sous la forme d'arbres de décisions.

2.3.2 Extensions du modèle BDI

Le modèle BDI reprend l'idée des mondes possibles, mais l'étend sous la forme de trois ensembles de mondes possibles. De plus, il introduit la notion de temps et d'arbre temporel.

Nous pouvons transformer un arbre de décisions et sa fonction de délibération en un modèle équivalent à trois ensembles de mondes possibles qui représentent les croyances, les désirs et les intentions comme des relations séparées sur des ensembles de mondes possibles. Cette transformation fournit une alternative pour les cas où les informations sur les probabilités et les *payoff* sont insuffisantes.

La transformation d'un arbre en un ensemble de mondes possibles se fait de la manière suivante [RG95]. Nous considérons un arbre complet où chaque chemin possible est représenté. Nous commençons par la racine, et empruntons les arcs jusqu'aux feuilles. Pour chaque arc émanant d'un noeud de possibilité, nous créons un nouvel arbre de décisions qui est identique à l'original à l'exception du fait que le noeud de possibilité est retiré et l'arc incident au noeud de possibilité est relié au successeur de ce noeud. Ce processus est répété récursivement jusqu'à ce qu'il n'y ait plus de noeud de possibilité. Il permet d'obtenir un ensemble d'arbres de décision où chaque arbre représente un état de l'environnement. Chacun de ces arbres représente un monde possible différent avec une certaine probabilité d'occurrence.

Le modèle des mondes possibles étendu par le modèle BDI correspond donc à trois ensembles de mondes possibles où chacun des mondes a une structure d'arbre. Le modèle BDI introduit également la notion temporelle. Ainsi, un monde est représenté sous la forme d'un arbre temporel (*time tree*).

La figure 2.3 est un exemple simple d'arbre temporel. Selon la syntaxe présentée dans l'annexe A, nous pouvons constater que l'arbre possède la propriété

inevitable($\Diamond q$) car la proposition q est vraie en un point de chaque branche de l'arbre du temps. Nous pouvons également dire *optional*($\Box r$) car la proposition r n'est vraie que sur une branche de l'arbre (*optional*) mais à chaque instant ($\Box$) de la branche.

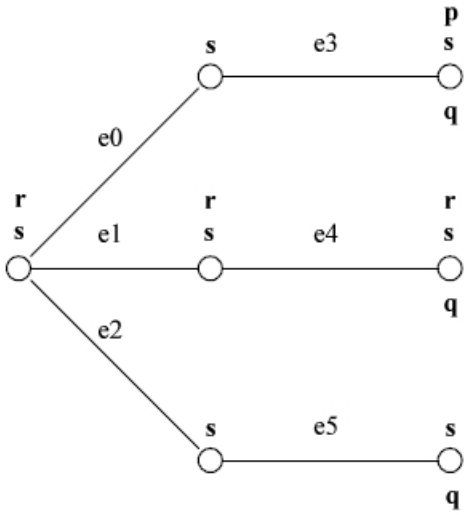


FIG. 2.3 – Exemple simple d'arbre temporel [RG91]

Un point particulier dans un monde, à un moment précis est appelé *situation*. Chaque situation est associée à un ensemble de mondes de croyances, un ensemble de mondes de désirs, et un ensemble de mondes d'intentions. Ces mondes représentent respectivement ce que l'agent croit être possible, ce qu'il désire causer, et ce qu'il a l'intention de causer. Les arcs des arbres temporels représentent des évènements. Les évènements transforment une situation particulière en une autre. Les branches en une situation représentent les choix possibles d'un agent à un moment précis.

Chaque monde de croyances est associé à un monde de désirs et un monde d'intentions. Mais l'inverse n'est pas vrai. En effet, même si un agent croit que certains faits sont inévitables, il n'est pas obligé de les adopter comme désirs ou comme intentions.

2.4 Architecture BDI des agents JACK

Le modèle d'agent BDI proposé par Jack a pour origine les travaux de Bratman [Bra87] ainsi que ceux de Rao et Georgeff [RG91] décrit dans l'annexe A. L'architecture BDI avait déjà été implémentée auparavant dans le *Procedural Reasoning System* (PRS) [GI88], et dMARS [dKLW98a].

Les éléments fondamentaux de la plate-forme Jack sont détaillés dans le chapitre 5. Nous nous contenterons d'expliquer ici les concepts de Jack liés au modèle BDI. Tout comme l'architecture globale que nous avons détaillée à la section 2.2, Jack utilise une architecture composée de capacités (*capabilities*), de croyances

(*beliefsets*), d'évènements (*events*), de plans, et d'objectifs. Cette architecture (figure 2.4) possède un ensemble de classes et de méthodes qui rendent possible une utilisation plus facile du raisonnement d'agent.

Les raisonnements des agents BDI Jack possèdent une ou plusieurs des caractéristiques suivantes :

- *Meta-raisonnement.* Il s'agit d'une technique d'écriture de plans sur la manière de sélectionner le plan le plus approprié étant donné une certaine situation.
- *Reconsidération des plans alternatifs en cas d'échec.* Si un plan échoue, cette technique autorise un système agents à (re)considérer d'autres lignes de conduite pour atteindre l'objectif.
- *Recalcul de l'ensemble des plans applicables.* Quand on veut traiter l'échec d'un plan, il est possible de créer un nouvel ensemble de plans applicables excluant tout plan déjà échoué.

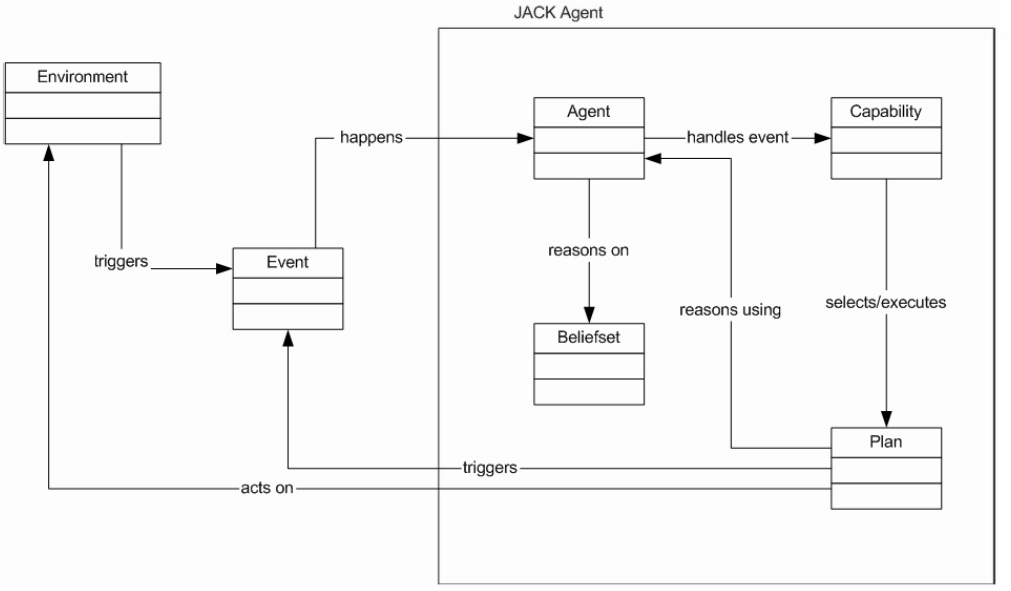


FIG. 2.4 – Vue globale de l'architecture BDI d'un agent Jack

Le concept de *capacité* permet de structurer les éléments de raisonnement d'un agent. Il simplifie le design, permet la réutilisation et l'encapsulation des fonctionnalités de l'agent. Une capacité est composée de croyances, d'évènements, de plans, de procédures personnalisées (en Java), ainsi que d'autres capacités.

Les *beliefsets* représentent les croyances de l'agent. Ils sont utilisés pour maintenir les croyances de l'agent sur le monde qui l'entoure. Ils ont été spécialement conçus pour être interrogés, et utilisent une structure logique. Ils ont été imaginés sur base du modèle relationnel, mais suivent des règles de programmation logique, similaires à celles du langage Prolog. La cohérence des *beliefsets* et les contraintes d'intégrité sont maintenues automatiquement. De plus, ils ont la capacité de générer automatiquement des évènements quand les croyances changent.

Les *évènements* décrivent une occurrence dans l'environnement à laquelle l'agent doit répondre en effectuant une action. Jack propose en particulier un évènement BDI. Celui-ci permet la conception d'agents aux comportements orientés objectifs et autorise l'agent à poursuivre un objectif à plus long terme. Un agent peut dès lors réunir un ensemble de plans pour répondre à un évènement donné, puis appliquer une heuristique qui choisit le plan le plus adapté et enfin agir.

Les classes d'évènements les plus importantes pour le raisonnement BDI d'un agent Jack sont présentées dans le tableau 2.1 :

Classe d'évènement	Description
`BDIFactEvent`	Il s'agit de la classe de base des évènements BDI Jack. Ils sont uniquement utilisés de manière interne par un agent.
`BDIMessageEvent`	Ces évènements sont reçus par un agent en provenance d'un autre agent.
`BDIGoalEvent`	Il représente un objectif que l'agent souhaite atteindre.
`PlanChoice`	Il s'agit d'un évènement utilisé de manière interne par un agent quand il doit choisir parmi plusieurs plans.

Tab. 2.1 – Classes d'évenements Jack [Sof04]

Les *plans* de l'agent sont analogues à des fonctions. Ils sont composés d'instructions que l'agent exécute en réponse à un évènement et de manière à atteindre un objectif (ou *désir*).

La classe `plan` décrit une séquence d'actions que l'agent peut prendre lorsqu'un évènement se produit. Un agent détermine si un plan est une réponse appropriée à un évènement en exécutant la méthode `relevant()`. Une fois que les plans appropriés ont été sélectionnés, il faut déterminer lesquels sont applicables en utilisant la méthode `context()` du plan. Cette méthode est une fonction logique propre au plan qui peut lier à ce plan les valeurs des membres logiques des *beliefsets*. Pour chaque lien, une instance de plan est générée. Si plusieurs instances de chaque plan approprié sont applicables, il faut sélectionner la plus appropriée, en postant l'évènement `PlanChoice`. Un agent exécute un plan en utilisant la méthode `body()` de ce plan. Elle est aussi appelée méthode de raisonnement. Si la méthode `body()` se termine, le plan réussit tandis que l'échec de cette méthode représente un échec du plan.

2.5 Conclusion

Le modèle de raisonnement BDI est basé sur des attitudes mentales. Si la sémantique de ce modèle est complexe, l'architecture est au contraire très simple. En particulier l'architecture BDI de la plate-forme Jack, composée de quelques classes Java, permet de modéliser simplement un raisonnement d'agents proche de celui des humains.

Nous mettons en pratique ce type de raisonnement dans notre étude de cas au chapitre 11.

Chapitre 3

Les étapes nécessaires à la construction d'un logiciel complexe

Sommaire

3.1 Introduction

Avant de nous consacrer dans les deux chapitres suivants aux notions de méthodologie et de plate-forme de développement que nous allons utiliser pour réaliser notre projet (chapitres 4 et 5), il serait bon de rappeler quelles sont les étapes indispensables à la construction de tout logiciel complexe et la qualité requise dans la mise en oeuvre de chacune de ces étapes.

Dans ce cadre, nous mettrons en évidence l'existence d'une nouvelle étape qui vient s'ajouter à la suite des trois étapes traditionnelles et qui se révèle pertinente dans la construction de système multi-agents.

Finalement, nous insisterons sur les qualités indispensables des différentes étapes et comment ces qualités se traduisent, se concrétisent au niveau pratique lors de la construction du système.

3.2 Les quatre étapes nécessaires à la construction d'un Système Multi-Agents

Les systèmes multi-agents sont des **logiciels complexes**. Elaborer de tels systèmes complexes requiert l'utilisation de méthodes d'ingénierie adéquates. Traditionnellement, l'ingénierie du software distingue trois étapes dans la construction de ces systèmes :

1. Analyse
2. Conception[1]
3. Développement

Dans leur article « *From Analysis to Deployment : a Multi-Agent Platform Survey* »[RD00], Pierre-Michel Ricordel et Yves Demazeau estiment que pour des systèmes tels les systèmes multi-agents cela n'est pas suffisant, et que dans la description du processus de création du système multi-agents[2] (SMA), du design de base jusqu'à l'exécution de l'application, il faut distinguer une quatrième étape qui suit l'étape de développement, appelée **déploiement**.

Ils affirment également que les frontières exactes entre les trois étapes classiques (analyse, design et développement) sont toujours sujettes à débat dans l'ingénierie du software. Puisqu'ils estiment que *« ces étapes sont juste des niveaux quantifiés le long d'un continuum conceptuel naturel »*, ils proposent d'utiliser les définitions suivantes pour ces quatre étapes :

Analyse Le processus de découverte, séparation et description du type de problème et du domaine environnant. En pratique, cela consiste à identifier le domaine d'application, les objectifs et fonctionnalités du futur système et les problèmes clés à résoudre.

Design Le processus de définition de l'architecture qui solutionne le problème. En pratique, cette étape consiste à spécifier la solution principale du problème, par exemple en utilisant AUML[3] ou toute autre méthodologie adaptée au SMA.

Développement Le processus de construction d'une solution fonctionnelle au problème. En pratique, cela consiste à coder la solution au moyen d'un langage de programmation particulier.

Déploiement Le processus d'application de la solution au problème réel dans le domaine donné. En pratique, le déploiement consiste à exécuter le logiciel sur un réseau d'ordinateurs, et alors, de maintenir et d'étendre ses fonctionnalités.

Les deux premières étapes sont certainement plus liées à des aspects *méthodologiques*, les deux suivantes à des aspects *techniques*.

[1] Nous utiliserons régulièrement le terme anglais *Design* souvent utilisé par les concepteurs francophones.

[2] *Multiagent System (MAS)* en anglais.

[3] Le site `http://www.auml.org` est dédié à l'*Agent Unified Modeling Language (AUML)*, une extension du célèbre UML. L'annexe D sera consacrée à la notation AUML.

3.3 Les quatre qualités indispensables des différentes étapes

Dresser les qualités significatives de chacune des quatre étapes de construction d'un software complexe n'est pas chose évidente. Par *qualité*, nous entendons un trait qui caractérise un aspect positif ou négatif dans la réalisation pratique d'une étape particulière. Quatre qualités s'appliquent à toutes les étapes et touchent à autant de pièges possibles du développement pratique. Ces quatre qualités sont :

- **Complétude** : le degré de couverture que la méthodologie ou la plate-forme fournissent pour cette étape. Sont concernés à la fois la quantité et la qualité de la documentation et les outils fournis.
- **Applicabilité** : le « scope » de l'étape c'est-à-dire en d'autres mots, l'étendue des possibilités offertes et les restrictions imposées par l'étape.
- **Complexité** : la difficulté de réaliser, de parachever l'étape. Nous parlons dès lors de compétences requises par l'utilisateur et de quantité de travail requise par la tâche.
- **Réutilisabilité** : la quantité de travail gagnée en réutilisant des travaux antérieurs.

3.4 Application des quatre qualités aux quatre étapes

Des quatre qualités appliquées aux quatre étapes de la construction d'un système multi-agents découlent 16 critères.

Analyse

Les critères d'analyse incluent :

- Complétude : La méthode d'analyse est-elle utile, et est-elle bien documentée ?
- Applicabilité : A quels domaines et problèmes cette méthode peut-elle être appliquée ?
- Complexité : La méthode d'analyse est-elle facile à comprendre et facile à appliquer ?
- Réutilisabilité : Existe-t-il une façon d'exploiter des analyses antérieures ou problèmes similaires ou de domaines similaires ? Y a-t-il des exemples d'analyse fournis ?

Design

Les critères de design incluent :

- Complétude : La méthode de design est-elle utile et bien documentée ? Existe-t-il des outils pour supporter le processus de design ?
- Applicabilité : Quel genre de SMA peut être conçu par cette méthode ?
- Complexité : La méthode de design est-elle facile à comprendre et à appliquer ?
- Réutilisabilité : Existe-t-il un moyen de réutiliser des designs antérieurs ? Y a-t-il des designs utiles fournis ?

Développement
Les critères de développement sont :
- Complétude : A quel point les outils de développement fournis sont-ils utiles ?
- Applicabilité : Certaines fonctionnalités sont-elles impossibles à implémenter avec les outils de développement ?
- Complexité : Les outils et langages de développement sont-ils faciles à employer ? A quel point le langage utilisé est-il populaire ?
- Réutilisabilité : Existe-t-il un support pour réutiliser le code ?

Déploiement
Les critères de déploiement incluent :
- Complétude : Un support est-il fourni pour déployer le SMA ?
- Applicabilité : L'outil de déploiement supporte-t-il la visualisation ou la maintenance en ligne ?
- Complexité : Les outils de déploiement sont-ils faciles à utiliser et à apprendre ?
- Réutilisabilité : Est-il possible d'intégrer dynamiquement un agent préalablement existant dans le nouveau système multi-agents sans aucune modification ?

D'autres critères peuvent également être considérés notamment lors du choix d'une plate-forme :
- Disponibilité : Y a-t-il une version d'évaluation, des clauses de confidentialité, le code source est-il disponible ? Combien cela coûte-t-il ?
- Support : Quels sont les futurs développements de cette plate-forme ? Cette plate-forme est-elle utilisée à grande échelle ?

3.5 Conclusion

En ce qui nous concerne, les étapes seront sans doute quelque peu modifiées, ou tout au moins dire adaptées, suite au choix d'une méthodologie particulière et/ou d'un environnement de développement spécifique mais il nous sera certainement utile de conserver ces quatre étapes à l'esprit pour mieux nous guider dans la réalisation de notre solution, notre application, tout en attachant un grand intérêt aux qualités requises lors de la réalisation de chacune de ces étapes.

Deuxième partie

Méthodologie et plates-formes des systèmes Multi-Agents

Chapitre 4

Prometheus, une méthodologie de développement de logiciel orientée agents

Sommaire

4.1 Introduction

Au chapitre 1, nous avons introduit les concepts importants des agents software. Sur base de ces concepts, nous allons tenter de répondre à la question suivante : *« Comment construire un système multi-agents ? »*. Naturellement, c'est ici qu'intervient la notion importante de **méthodologie de développement de software**.

Pour réaliser notre objectif qu'est la construction du système multi-agents, nous avons besoin d'un processus qui permette de diviser la construction dudit

système en plus petites étapes que nous suivons afin de spécifier, de concevoir et de construire notre système multi-agents. Ce besoin, nous allons le satisfaire grâce à **Prometheus**, une nouvelle méthodologie de développement software orientée agents ou méthodologie AOSE pour « *Agent Oriented Software Engineering* ». Dans le présent chapitre, nous présentons les méthodologies actuelles (orientées objets et orientées agents), la justification de notre choix de méthodologie et la présentation détaillée de celle-ci.

4.2 Les caractéristiques d'une méthodologie de développement

En plus d'étapes de haut niveau tel que « *spécifier le système* » ou encore « *identifier les objectifs du système* », une méthodologie pratique au niveau de son utilisation se doit de fournir des directives[1] détaillées expliquant comment réaliser chaque étape [WP04]. Concrètement, ces *guidelines* prennent la forme d'heuristiques ou d'exemples : il est en effet très difficile d'avancer des règles strictes à suivre et l'application d'une méthodologie ou plutôt les prises de décision à propos du design résultent souvent de compromis.

Au fur et à mesure de la progression de l'analyse, des documents[2] englobant des informations concernant le système sont produits. Ces *artifacts* de design ou documents d'analyse sont souvent spécifiés au moyen de *notations* formelles ou plus couramment semi-formelles et qui peuvent être graphiques ou textuelles.

En résumé, une méthodologie de développement de software devrait fournir un *processus* muni de *guidelines* détaillées (incluant heuristiques et exemples) et des *notations* utilisées pour décrire les *artifacts* de design.

4.3 Pourquoi introduire une nouvelle méthodologie ?

La question peut sembler inopportune mais après réflexion, elle est en réalité fondée sur une observation : à l'heure actuelle, bon nombre de méthodologies de développement existent déjà (nous pouvons assurément parler de plusieurs dizaines), certaines ayant fait leurs preuves plus que d'autres. Pourquoi ne pas tout simplement utiliser l'une d'entre elles ?

Comme nous avons pu nous en rendre compte auparavant, les agents sont souvent considérés par les experts comme une suite logique du paradigme de programmation objets possédant une multitude de points communs avec eux mais avec un niveau d'abstraction plus élévé. Nous pourrions donc tout à fait imaginer sélectionner une méthodologie parmi les nombreuses méthodologies orientées objets existantes et appliquer celle-ci à notre cas de figure. En effet, nombreuses

[1] *Guidelines* en anglais.
[2] Le terme anglais *artifacts* est souvent utilisé.

sont les méthodologies orientées objets qui furent considérablement étudiées et développées et qui le sont toujours à l'heure actuelle.

Cependant, nous avons également remarqué que les concepts de base des agents diffèrent des objets. C'est la raison pour laquelle nous allons utiliser une nouvelle méthodologie adaptée au domaine multi-agents. En effet, bien que les agents et les objets affichent des similitudes, les différences restent significatives. Une analyse et un design orientés objets sont possibles dans le cas de système multi-agents mais le couple n'est pas naturel et le résultat fera sans doute mauvais usage des propriétés qui caractérisent les agents.

Par exemple, un aspect important des agents est qu'ils sont ou plutôt peuvent être *proactifs* c'est-à-dire qu'ils tentent de réaliser des objectifs et ce de façon permanente, jusqu'à atteindre le succès. Cette persistance est d'autant plus intéressante qu'elle rend les agents robustes. Puisque l'échec de certaines de ses actions (nous devrions plutôt utiliser le terme *plans* grâce auxquels ils réalisent ses objectifs) est tout à fait possible (l'agent évolue dans un environnement changeant), l'agent doit pouvoir se rétablir de toute situation d'échec, s'en relever, agissant de la sorte comme un agent **robuste**.

La proactivité chez les agents est cruciale et doit apparaître lors des phases d'analyse, la méthodologie de développement devant offrir cette possibilité. Le problème est que la proactivité et plus précisément les objectifs que les agents désirent réaliser ne sont pratiquement pas exprimables avec des méthodologies orientées objets. A l'opposé, les méthodologies de développement de logiciel orientées agents et plus précisément **Prometheus** permettent de modéliser des agents proactifs et leurs objectifs associés. Il en va de même des autres concepts spécifiques au paradigme agents.

Prometheus et les méthodologies orientées agents diffèrent des méthodologies orientées objets par le support qu'elles offrent des concepts spécifiques aux agents :

- l'apport de méthodes permettant de définir des *types* d'agents ;
- la considération des messages entre agents comme *composants*, ce qui permet à ceux-ci d'être traités par de multiples plans, autorisant ainsi les agents à atteindre les qualités de flexibilité et robustesse ;
- la distinction entre *perception* et *action*, les premiers étant l'interface entre l'agent et son environnement, les seconds ce qu'entreprend un agent après la perception de toute modification de son environnement ; et par,
- la distinction entre composants *passifs* (données et croyances) et composants *actifs* (agents, capacités et plans).

Nous voyons bien l'inadéquation des méthodologies orientées objets et leur inefficacité c'est pourquoi nous nous tournons définitivement vers les méthodologies orientées agents auxquelles **Prometheus** appartient. La dernière décennie a vu l'apparition d'une multitude de ces nouvelles méthodologies. Au cours des points 4.4 (page 42) et 4.5 (page 42) nous allons nous intéresser aux plus célèbres, les étudier brièvement, les comparer et justifier notre choix final, **Prometheus**.

4.4 Les méthodologies orientées agents

> *«One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems.»* [LMP03]

Afin d'étayer ces propos tenus par Michael Luck, Peter McBurney et Chris Preist, nous affirmons qu'un large éventail de méthodologies orientées agents ont vu le jour au cours de ces dernières années mais toutes ne sont pas encore complètement « abouties ». Quelques-unes d'entre elles sont toutefois le fruit d'un travail de longue haleine ; maintes fois éprouvées, elles furent constamment améliorées, développées pour atteindre des hauts niveaux de précision, de facilité d'utilisation et de complétude. Parmi ces méthodes nous retenons les noms suivants : **Gaia**, **Tropos**, **MaSE** et encore **Prometheus**.

Naturellement, cette liste n'est pas exhaustive et d'autres noms pourraient y être ajoutés. Nous avons volontairement limité le nombre de méthodologies parmi lesquelles nous allons sélectionner notre méthodologie afin de construire notre système multi-agents. Pour réaliser cette sélection, nous allons en premier lieu décrire chacune d'elles. Ensuite, nous les comparerons afin d'étudier celles qui semblent les plus abouties, simples d'utilisation et faciles d'apprentissage ou encore celles qui offrent des outils de support. Ces trois critères ne sont qu'une petite partie de l'ensemble des critères selon lesquels vont être comparées les méthodologies de développement. Cependant, puisqu'il s'agit pour nous de construire notre premier système multi-agents, les critères que nous avons listés ci-dessus seront relativement importants à nos yeux.

4.5 Pourquoi choisir Prometheus ? Une comparaison des méthodologies orientées agents actuelles

Khanh Hoa Dam et Michael Winikoff proposent une excellente comparaison [DW03] des méthodologies **MaSE**, **Prometheus** et **Tropos**. Elle s'intéresse à quatre éléments importants : *les concepts*, *le langage de modélisation*, *le processus* et *les aspects pragmatiques*. En annexe B, nous avons ajouté une description et quelques commentaires supplémentaires à propos de la méthodologie Gaia puisqu'elle ne figure pas dans la comparaison de référence. Une étape cruciale dans le travail de comparaison est de comprendre les relations entre les différentes méthodologies, leurs similitudes et mieux encore, de comprendre les forces, faiblesses et domaines d'applicabilité de chacune. Le lecteur désireux de connaître chacune des méthodologies en détail pourra également trouver en annexe B une description des méthodologies **Tropos**, **MaSE** et **Prometheus**. Dans le suite de ce chapitre, nous allons dans un premier temps expliquer les différents critères de comparaison utilisés puis dans un deuxième temps, présenter les résultats de la comparaison qui nous permettront d'effectuer un choix judicieux.

4.5.1 Les critères de comparaison

Nous avons vu au paragraphe 4.2 (page 40) les caractéristiques que devrait posséder une bonne méthodologie. Revenons maintenant sur les critères sur base desquels nous pouvons comparer nos méthodologies puisque, pour rappel, nous voulons justifier notre choix de méthodologie de développement de logiciel au cours de ce chapitre.

Ainsi, comme nous l'avions déjà rapidement évoqué, notre comparaison va s'effectuer sur les critères suivants :

Les concepts : Basé sur une recherche dans la littérature, Khanh Hoa Dam et Michael Winikoff, les auteurs de la comparaison [DW03], ont remarqué qu'un ensemble de concepts significatifs orientés agents est fréquemment mis en évidence. Cet ensemble inclut la définition des agents, leurs caractéristiques comme l'autonomie, l'adaptabilité, les notions mentales (telles que *beliefs*, *desires* et *intentions*), les relations et la communication entre les agents ou encore d'autres concepts comme les objectifs, les rôles et capacités des agents ainsi que les perceptions, actions et événements. Il s'agit donc de voir jusqu'à quel point chacune des méthodologies *supporte ces différents concepts* ou encore jusqu'à quel point chacune *supporte la construction d'agents qui possèdent ces attributs.*

Le langage de modélisation : Si les concepts orientés agents sont la base de toute méthodologie de développement de software, le langage de modélisation est le coeur du problème dans la construction des différents designs en termes de ces concepts. Un langage de modélisation typique consiste en trois composants principaux : les *symboles* graphiques ou textuels (permettant de représenter les graphiques), la *syntaxe* et enfin la *sémantique.* Il est important que le langage de modélisation permette de modéliser le système selon différentes vues telles que comportementale, fonctionnelle ou encore structurelle.

Le processus : Il s'agit également de mettre l'accent sur la série d'activités et d'étapes accomplies comme partie d'un cycle de vie complet d'un logiciel. Ces activités et étapes forment le processus qui assiste les analystes, les développeurs et les managers du système dans le développement du software. Selon Thomas W. Malone et Kevin Crowston [MC94], une méthodologie idéale devrait couvrir six étapes, qui sont la modélisation de l'entreprise, l'analyse du domaine, la spécification des besoins, le design, l'implémentation et les tests. Toute méthodologie qui couvre tous les aspects du développement est bien entendu susceptible d'être choisie puisqu'elle est *consistante* et *complète.*

Les aspects pragmatiques : Le choix d'une méthodologie dépend également des aspects pragmatiques. Elle peut être évaluée en se basant sur les problèmes de management et de technique qui pourraient survenir. Le *critère de management* devrait considérer le support que la méthodologie fournit lorsqu'elle est adoptée. Il inclut :

– le *coût* qu'implique la sélection de la méthodologie c'est-à-dire le coût d'ac-

quisition de la méthodologie, du ou des outils de support et enfin le coût de formation pour l'exploiter pleinement ;
- la *maturité* qui concerne les ressources disponibles pour supporter la méthodologie (par ex. la documentation, les formations, les services de consultance, etc.) et la *disponibilité d'outil(s) de support* ; et enfin,
- l'*histoire* de la méthodologie c'est-à-dire les conditions dans lesquelles la méthodologie a déjà été éprouvée (mise en pratique « sévère » en entreprise ou simples prototypes de démonstration).

Le deuxième critère, le *critère technique*, s'intéresse au(x) domaine(s) de software spécifique(s) au(x)quel(s) est adapté la méthodologie. Plus la méthodologie couvre un large éventail de domaines software, plus elle est susceptible d'être choisie.

4.5.2 La comparaison des méthodologies

Venons-en finalement au vif du sujet en présentant les résultats de la comparaison des différentes méthodologies. Les résultats sont basés sur les réponses aux questionnaires qu'ont reçu les auteurs des méthodologies (questionnaire traitant les quatre critères que nous avons vu au point 4.5.1) mais aussi le fruit d'une collaboration avec des étudiants qui ont eu l'occasion de donner, après les avoir utilisées dans des projets particuliers, des commentaires sur les différentes méthodologies. Ces résultats sont résumés ci-dessous dans les tables 4.1, 4.2, 4.3 et 4.4 et seront discutés en traitant chacun des quatre critères séparément.

Les concepts (Tableau 4.1) : En ce qui concerne les différents concepts spécifiques aux agents, le niveau de support des trois méthodologies est globalement bon (de moyen à élevé). Prometheus et Tropos supportent très bien l'utilisation des *attitudes mentales* (*beliefs*, *desires* et *intentions*) en permettant de modéliser la structure et le fonctionnement internes des agents. Aussi bien MaSE que Prometheus supportent la modélisation des aspects dynamiques du système et gèrent relativement bien les protocoles. Tropos ne fournit pas de support complet pour les protocoles, ou pour la modélisation des dynamiques du système excepté quelques précisions au niveau du design détaillé. Seul petit bémol concernant Prometheus, les étudiants rencontrent parfois quelques difficultés avec la terminologie et confondent des concepts comme perception, *incident*, *trigger* et événement.

Le langage de modélisation (Tableau 4.2) : Les participants estiment que les systèmes de notation respectifs des différentes méthodologies sont clairs et raisonnablement bien définis (que ce soit au niveau syntaxique ou sémantique) et qu'ils sont assez faciles à utiliser. Les possibilités de traçabilité ne sont pas non plus remises en cause puisque tout à fait bien supportées dans les trois cas. Seule ombre au tableau, la vérification de la consistance dont les niveaux de support ne sont pas les mêmes et différents d'une méthodologie à l'autre. MaSE et Prometheus la supportent très bien contrairement à Tropos. Pour en terminer avec ce critère, la satisfaction est de mise auprès des étudiants qui apprécient notam-

Concepts & Propriétés	MaSE	Prometheus	Tropos
Autonomie	H/M/NSP	H/PA/H	H/M/M
Attitudes mentales	B/M/H	H/M/H	H
Proactivité	H/M/H	H/M/NSP	H
Réactivité	M	H/M/NSP	H/B/NSP
Concurrence	H/M/H	M/B/NSP	H/M/H
Travail en équipe	H/M/H	N/B/PA	H/H/M
Protocoles	H	M/H/M	PA/M/M
Situation	M/B/H	H	H
Concepts clairs	SP/P/P	P/P/C	SP/P/N
Concepts surchargés	P/N/SP	N	SC/N/C
Orienté agents	SP/P/P	SP	SP/P/SP

TAB. 4.1 – Les concepts et propriétés des différentes méthodologies [DW03]. Notations : B pour Bas, M pour Moyen, H pour Haut, NSP pour Ne Sais Pas, SC pour Sévèrement Contre, C pour Contre, PA pour Pas Applicable, N pour Neutre, P pour Pour, SP pour Sévèrement Pour, _ pour pas de réponse. Les deux premières données dans chaque colonne proviennent des développeurs de la méthodologie, la troisième de l'étudiant. Une seule entrée dans une colonne indique que les trois réponses sont identiques.

ment Prometheus et son diagramme de vue d'ensemble du système[3] jugé très utile.

Modélisation & Notations	MaSE	Prometheus	Tropos
Statique & Dynamique	SP/P/P	SP/P/P	N/P/P
Syntaxe définie	P/P/SP	SP/P/P	N/P/P
Sémantique définie	P/SP/SP	P	SP/P/P
Notation claire	P	SP/P/P	SP/P/N
Facile à utiliser	SP/P/P	P/N/P	SP/P/N
Facile à apprendre	N/N/P	SP/PA/SP	SP/N/P
Différentes vues	N/N/P	P/P/SP	SP/P/N
Langage adéquat & expressif	SP/N/N	P	SP/P/N
Traçabilité	P/SP/SP	P	P/N/P
Vérification de la consistance	SP/P/SP	SP/P/P	_/P/C
Raffinement	SP/P/P	SP	SP/P/C
Modularité	SP/P/P	SP/SP/P	SP/P/N
Réutilisabilité	N/SP/P	N/P/N	_/C/C
Modélisation hiérarchique	PA/P/P	SP/P/P	SP/P/C

TAB. 4.2 – La modélisation et les notations des différentes méthodologies [DW03]. Notations : B pour Bas, M pour Moyen, H pour Haut, NSP pour Ne Sais Pas, SC pour Sévèrement Contre, C pour Contre, PA pour Pas Applicable, N pour Neutre, P pour Pour, SP pour Sévèrement Pour, _ pour pas de réponse.

[3] *System Overview Diagram* en anglais.

Le processus (Tableau 4.3) : Au point de vue du cycle de vie, toutes les méthodologies couvrent les phases de spécification des besoins, de design architectural et de design détaillé. Les réponses des étudiants à ce sujet sont très encourageantes. Ils estiment que ces phases d'analyse sont très bien documentées et sont accompagnées d'exemples concrets et même d'heuristiques. Cela les aide considérablement à passer d'une pensée orientée objets à une pensée orientée agents. Par contre, la phase d'implémentation est de façon surprenante très peu supportée. Tropos explique brièvement le passage à la plate-forme JACK mais manque d'exemples, d'heuristiques ou même de discussion de problèmes fréquemment soulevés. Seuls MaSE et Prometheus mentionnent la phase de test/debugging mais, pour le premier, le point jusqu'auquel il la supporte n'est pas très clair, et pour le deuxième cela fait encore partie d'un projet de recherche, pas encore intégré dans les outils mis à disposition des développeurs.

Processus	MaSE	Prometheus	Tropos
Spécification des besoins	SPEH	SPEH	SPE
Conception architectural	SPEH	SPEH	SPE
Conception détaillée	SPEH	SPEH	SPE
Implémentation	SEH/SPE/S	SPEH/S/r	SE/SPE/SPEH
Test & Débogage	SPE/r/r	SPEH/S/r	r
Deploiement	SE/SPE/SPEH	r	r
Maintenance	r/SPE/r	r	r

TAB. 4.3 – Les processus des différentes méthodologies [DW03]. Notations : S pour Etape (Stage) mentionnée, P pour Processus donnés, E pour Exemples donnés, H pour Heuristiques donnés, r pour rien.

Les aspects pragmatiques (Tableau 4.4) : Selon les réponses obtenues des auteurs des méthodologies, MaSE et Prometheus sont destinés à des étudiants et des développeurs travaillant dans le milieu industriel alors que Tropos serait plus adéquat pour des experts du domaine des agents. Il est évident que mesurer la complexité d'une méthodologie n'est pas chose aisée. La disponibilité de documents (publications dans des journaux spécialisés, articles de conférence, guide d'utilisateur, etc) reste limitée. Aucune des méthodologies ne semble offrir des assurances de qualité ou des « *guidelines* » pour l'estimation des coûts. Finalement, la disponibilité d'outil(s) de support varie également. MaSE et Prometheus sont bien supportés contrairement à Tropos qui n'offre qu'un support faible.

A la lecture de cette comparaison, nous avons choisi de donner notre préférence à **Prometheus** comme méthodologie de développement de notre software orienté agents et ce, pour les raisons principales suivantes :

- elle met en oeuvre le développement par itération qui permet d'une part, d'apporter des améliorations, des modifications et des nouveux éléments à chaque itération et d'autre part, de ne pas devoir tout définir de façon parfaite lors de la première itération ;

Aspects pragmatiques	MaSE	Prometheus	Tropos
Qualité	N/C/P	P/N/N	C/P/_
Estimation des coûts	_/C/SP	C/C/N	C/N/_
Gestion des décisions	_/C/SP	SC/N/_	SP/P/_
apps	21+	6-20	1-5
Real apps	non	non	non
Utilisé par les non-créateurs	oui	oui	oui/non/non
Domaine spécifique	non	non	oui/non/non
Echelonnable	_/N/N	N/P/N	N/N/_
Distribué	_/SP/SP	SP/P/N	N/P/_

TAB. 4.4 – Les aspects pragmatiques des différentes méthodologies [DW03]. Notations : B pour Bas, M pour Moyen, H pour Haut, NSP pour Ne Sais Pas, SC pour Sévèrement Contre, C pour Contre, PA pour Pas Applicable, N pour Neutre, P pour Pour, SP pour Sévèrement Pour, _ pour pas de réponse.

- elle supporte très bien l'utilisation d'attitudes mentales comme le modèle BDI que nous retrouvons désormais dans plusieurs plates-formes de développement ;
- elle automatise la vérification de la cohérence et de la consistance des données aussi appelée procédure de *« cross-checking »* grâce au PDT (*Prometheus Design Tool*) ;
- elle constitue un bon choix de méthodologie pour un premier développement et qui plus est pour des étudiants (à l'heure actuelle, elle est enseignée depuis plusieurs années au sein de la RMIT University à Melbourne) ; et enfin,
- elle a été développée en Australie, terre d'accueil de notre stage, ce qui rend les contacts plus aisés en cas de nécessité.

4.6 Les caractéristiques détaillées de Prometheus

Après l'avoir analysée de manière globale au point B.3 (page IX), nous allons brièvement rappeler ce qu'est Prometheus puis définir cette méthodologie de façon beaucoup plus précise. Comme nous avons déjà pu le voir, la méthodologie Prometheus définit un processus détaillé dans le but de spécifier, de construire le design, d'implémenter et tester/debugger les systèmes orientés agents [WP04]. Elle est associée à une série « *d'artifacts* », graphiques ou textuels, produits tout au long de l'analyse, de la conception et de la construction du système et qui permettent la mise sur papier des concepts agents que nous avons notamment expliqués au cours des chapitres 1 et 2.

Comme nous l'avons déjà énoncé, la méthodologie *Prometheus* consiste en trois phases que nous retrouvons dans la figure 4.1.

1. La *phase de spécification du système* se concentre sur l'identification des objectifs, des fonctionnalités basiques, des *inputs* (perceptions) et *outputs* (actions).

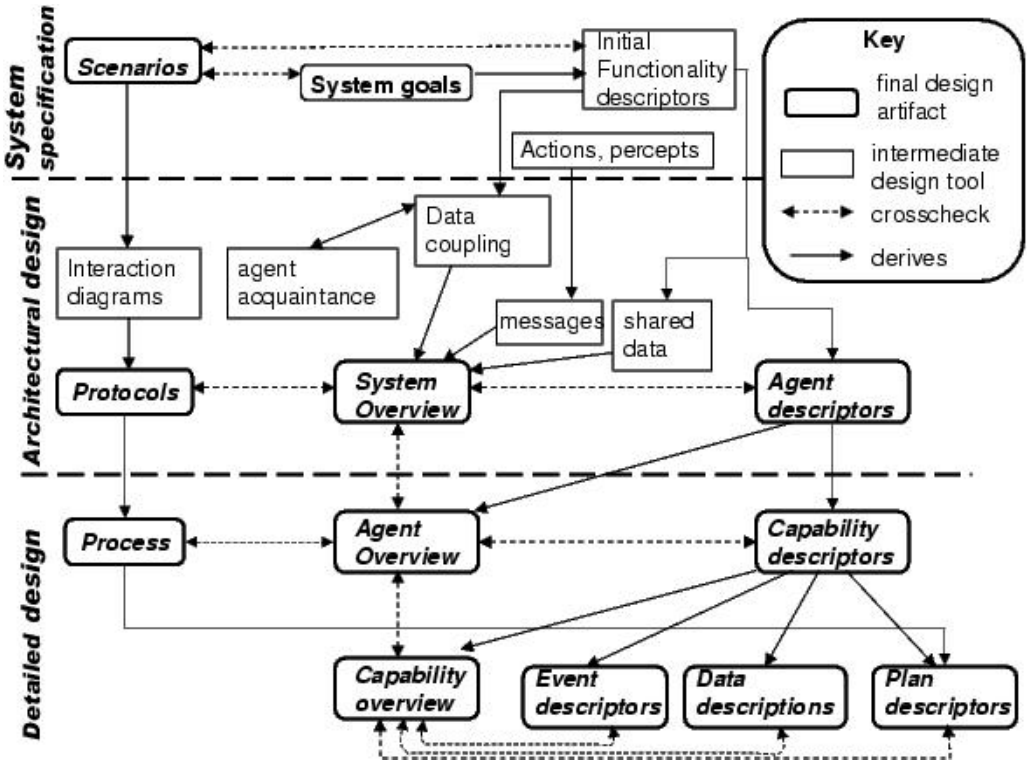


FIG. 4.1 – Les phases de la méthodologie Prometheus [WP04]

2. La *phase de design de l'architecture* utilise les outputs de la phase précédente pour spécifier des types d'agents et déterminer la manière dont ils interagissent.
3. La *phase de design détaillé* s'intéresse aux fonctionnements internes de chaque agent et comment il va accomplir ses tâches dans le système global.

La quatrième phase est bien entendu la phase d'implémentation qui n'est pas présente dans la figure 4.1 puisqu'elle dépend de la plate-forme d'implémentation choisie. Attardons-nous quelque peu sur ces trois phases afin d'en dégager les éléments clés.

4.6.1 Spécification du système

La phase de spécification du système se concentre sur :
- l'identification des **objectifs du système** ;
- la spécification de **scénarii d'utilisation** illustrant les opérations du système ;
- l'identification des **fonctionnalités** de base du système ; et enfin,
- la spécification de l'interface entre le système et ses environnements en termes d'**actions** et de **perceptions**.

L'identification des objectifs consiste à découvrir et spécifier les objectifs, les lister et les décrire afin de savoir ce que le système a besoin de faire.

Les scénarii permettent de représenter les opérations du système. Plus concrets que la spécification des objectifs, ils sont plus « accessibles » et faciles à visualiser. Ces scénarii sont assez proches des scénarii rencontrés dans les méthodologies orientées objets même s'ils peuvent différer sur quelques points plus précis.

Les fonctionnalités du système se construisent sur base des objectifs ou plutôt des regroupements ou agrégats de ceux-ci avec les données, les perceptions et les actions.

Enfin, l'environnement dans lequel interviennent les agents est défini en termes d'actions et de perceptions.

4.6.2 Design de l'architecture

La phase de design de l'architecture consiste à :
- déterminer les types d'agents qui vont être implémentés et écrire les *descripteurs*[4] associés ;
- définir la structure globale du système par un diagramme de vue d'ensemble du système ;
- définir le comportement du système par des diagrammes et des protocoles d'interaction.

La définition des différents types d'agents est bien entendu le produit principal de cette phase. Un type d'agents est construit en groupant plusieurs fonctionnalités qui ont des caractéristiques communes de couplage et de cohésion entre elles. Chaque type d'agents est décrit par un descripteur d'agent.

Un diagramme de couplage de données est également créé ainsi qu'un diagramme de connaissance des agents (ou *agent acquaitance diagram*) qui permet de représenter les relations existantes entre agents.

Finalement, à tout cela viennent s'ajouter une spécification des messages inter-agents et une représentation des données partagées permettant ainsi d'obtenir le diagramme de vue d'ensemble du système.

4.6.3 Design détaillé

La phase de design détaillé s'attache à étudier la structure et le fonctionnement interne de chaque type d'agents et la manière par laquelle celui-ci réalise ses tâches.

Dans un premier temps, les agents sont redéfinis en termes de capacités donnant ainsi naissance au diagramme de vue d'ensemble de l'agent, aux descripteurs de capacités et aux spécifications de processus.

Dans un deuxième temps, les événements et les plans associés qui traitent ces événements sont décrits.

Dans un dernier temps, la spécification des algorithmes dans chaque plan et des données et événements associés se retrouvent sous la forme de descripteurs de

[4] *Descriptor* en anglais, il s'agit des documents d'analyse rédigés permettant de décrire les différentes entités du système.

plans, de données et d'événements.

Ces derniers documents, fournissant les détails nécessaires, permettent d'envisager un choix d'implémentation. Dans notre cas, nous supposons que les plans sont exécutés en réponse à des événements et qu'il est tout à fait possible d'associer plusieurs plans à un événement, le choix du plan étant fonction de l'environnement du moment c.-à-d. des connaissances de l'agent mais aussi des caractéristiques de l'événement. Cette hypothèse correspond à une classe particulière de plates-formes d'implémentation basés sur le modèle BDI telles que la plate-forme JACK.

4.6.4 Directives d'utilisation, développement itératif, outil de support et cas d'utilisation

Directives d'utilisation

En tant que méthodologie de développement, Prometheus devrait être utilisée comme un *guide* dans la construction de toute application orientée agents. Cela n'implique pas un suivi strict, tête baissée et sans adaptation de la méthodologie. En effet, les concepteurs ont développé la méthodologie pour qu'elle remplisse certains objectifs et qu'elle soit suffisamment complète tout en la gardant bien de devenir trop complexe ou trop spécifique. Il est donc tout à fait normal que quelques cas pratiques ne requièrent pas un suivi à la lettre de chaque étape : par exemple, soit le problème est très simple et des concepts comme les capacités des agents ne sont pas présents dans la solution finale soit la conception et l'analyse engendrent de la redondance d'informations dans certaines phases ce qui est tout à fait inutile.

Quoi qu'il en soit, si la méthodologie est suffisamment complète au point de pouvoir accompagner tout novice dans ses premières démarches de développement, elle est aussi adaptable (elle peut être modifiée), adaptative (elle facilite la modification) et le concepteur averti et expérimenté pourra aisément passer au-delà de certaines phases ou étapes qui, à ses yeux, sont sans intérêt ou quelque peu redondantes.

Développement itératif et Outil de support

Désormais, s'il est une caractéristique que l'on peut retrouver dans la plupart des nouvelles méthodologies de développement, que ce soit orientées objets ou orientées agents, c'est bien le **développement par itération**. Contrairement aux anciennes méthodologies comme le modèle *waterfall*, il ne s'agit plus de tout définir de façon parfaite à la première mais aussi unique itération, mais bien de réaliser plusieurs itérations et d'apporter des améliorations, modifications et nouveaux éléments à chacune d'entre elles. Les étapes classiques telles que la spécification des besoins, le design de haut niveau, le design détaillé, l'implémentation ou encore les tests sont toujours bien présentes mais ces activités ne doivent pas être réalisées en séquence en une seule fois.

En général, lors de la ou des premières itérations, le concepteur s'attache à spécifier les besoins puis se charge des autres étapes lors des itérations suivantes. Petit bémol, le fait de modifier un design existant peut introduire des inconsistances. Pour éviter cela, la pratique du *cross-checking*[5] est de rigueur. Réalisée manuellement, elle constitue une tâche fastidieuse et est une porte ouverte aux erreurs. C'est pourquoi l'existence d'un outil de support réalisant ce *cross-checking* de façon automatique est une aide non-négligeable.

Cas d'utilisation

Bien évidemment, tous les logiciels ne peuvent pas être considérés, modélisés et construits en tant qu'agents. Parfois, un système hybride est le plus efficace, les agents côtoyant des composants plus traditionnels (par exemple orientés objets). De façon générale, les agents doivent être utilisés quand ils constituent des réponses plus naturelles aux problèmes et offrent des bénéfices et avantages. Ainsi, un composant qui répond à une majorité des questions suivantes de façon positive devrait sans aucun doute être considéré comme un agent :

- est-il autonome ?
- a-t-il des objectifs précis ?
- réalise-t-il plusieurs actions en même temps ?
- ce qu'il entreprend dépend-il de l'état de son environnement ?

La méthodologie Prometheus sera donc d'autant plus efficace qu'il s'agit pour elle de modéliser des composants qui seront construits comme des agents. Il est cependant intéressant de noter qu'il est possible d'adapter les différents « *artifacts* » de manière à intégrer des composants orientés objets à la modélisation.

[5] Procédure de vérification de la cohérence et de la consistance des données

Chapitre 5

JACK, une plate-forme de développement orientée agents

Sommaire

5.1 Introduction

A travers le chapitre 4, nous avons pu constater que l'émergence du nouveau paradigme de programmation orienté agents a donné lieu à l'élaboration de plusieurs méthodologies et architectures pour la modélisation des systèmes multi-agents. Nous en avons profité pour étudier les caractéristiques importantes des méthodologies adaptées au développement orienté agents.

Le concept de programmation orientée agents est une idée très intéressante et les méthodologies développées fournissent des patrons théoriques pour la modélisation des SMA[1]. Cependant, les systèmes à base d'agents spécifiés à partir de

[1] Pour rappel, Systèmes Multi-Agents.

ces méthodologies sont souvent *difficiles à implémenter directement* dans des langages de programmation standards comme Java ou C++ [GD02]. L'intensification des activités dans le domaine des agents a donc résulté en une pléthore d'outils, d'environnements, de plates-formes offrant des services pour le développement de SMA [AGK+01].

Après avoir opté pour la méthodologie Prométheus afin d'analyser notre application, nous allons donc, grâce au chapitre présent, approfondir nos connaissances dans ces plates-formes pour finalement désigner la plate-forme optimale pour la réalisation (le développement) de notre projet parmi un ensemble de plates-formes. Nous nous limiterons volontairement aux plates-formes les plus couramment citées et utilisées par les plus avertis.

En clair, tout comme nous l'avons fait pour les méthodologies de développement, nous allons clarifier le concept de plate-forme de développement de SMA en mettant notamment en avant les caractéristiques essentielles que devrait contenir un environnement complet de développement de systèmes multi-agents. Nous réaliserons une comparaison des plates-formes puis nous justifierons notre choix de plate-forme de développement, Jack, en décrivant celle-ci plus en détail.

5.2 Les environnements de programmation orientée agents

5.2.1 Les caractéristiques essentielles

Dans leur étude comparative d'outils et d'environnements de programmation orientée agents [GD02], Tony Garneau et Sylvain Deliste mettent l'accent sur les caractéristiques essentielles que devraient posséder un environnement complet de développement de systèmes multi-agents. Selon eux, des **lacunes persistent** et des **efforts devraient être entrepris** pour y remédier afin de développer des outils **utiles** et **applicables** dans la pratique.

Les caractéristiques essentielles d'un outil de développement orienté agents sont les suivantes :

- la mise à disposition d'un support simple pour le déploiement inter-machines ;
- la mise à disposition d'un support pour les bases de données ;
- la diminution par l'utilisation de l'outil, de l'effort nécessaire à l'implémentation et la quantité de code à écrire ;
- un niveau d'abstraction suffisant pour permettre aux personnes moins expérimentées en programmation orientée agents de créer facilement des SMA sans connaître tous les détails d'implémentation ;
- un degré de latitude laissée aux plus expérimentés leur permettant d'accéder et d'interagir directement avec les différents composants du système (l'outil doit ajouter un niveau d'abstraction à la programmation sans toutefois qu'il devienne un obstacle pour les programmeurs) ;
- l'extensibilité du code ;

- les développeurs n'ont pas à se soucier de l'implémentation du système de communication et des protocoles utilisés pour transporter les messages ;
- un utilitaire de débogage, une interface utilisateur facilitant le développement et un générateur automatique de code source ; et enfin,
- une documentation appropriée.

5.2.2 Objectifs des plates-formes de développement

Les critères d'évaluation que nous choisirons pour effectuer notre comparaison sont directement liés aux objectifs qui devraient être atteints pour considérer les outils comme étant des environnements de développement de SMA. Bien entendu, il se peut que les objectifs diffèrent d'une plate-forme de développement à l'autre mais il existe un ensemble nécessaire d'objectifs à atteindre.

Les objectifs indispensables d'un environnement de développement de SMA sont :

- accélérer le développement et diminuer l'effort de programmation ;
- abstraire les mécanismes de communication, d'interaction et de coordination ;
- permettre l'implémentation de systèmes relativement complexes ;
- permettre une bonne extensibilité du code ; et,
- fournir un support pour le déploiement (et l'exécution) des systèmes.

5.3 Comparaison des plates-formes actuelles

5.3.1 Choix des plates-formes de développement

C'est à Pierre Michel Ricordel et Yves Demazeau que nous devons *« From Analysis to Deployment : a Multi-Agent Platform Survey »* [RD00], une étude comparative de plates-formes multi-agents sur laquelle nous allons nous baser pour réaliser notre comparaison et effectuer notre choix final.

Naturellement, examiner des plates-formes et outils disponibles qui s'avèrent être actuellement les plus mauvais n'aurait pas grand intérêt ; c'est pourquoi il faut attacher de l'importance à la sélection des plates-formes à évaluer. Dans ce sens, et pour faire le lien avec les caractéristiques essentielles mises en avant au point 5.2.1, les environnements de développement que nous étudierons auront en commun :

- d'être « populaires » et régulièrement mis-à-jour (correction de « *bugs* » et nouvelles fonctionnalités disponibles) ;
- d'être fondés, de reposer sur des modèles théoriques bien connus ;
- d'être développés en suivant des standards de qualité comparables à des standards industriels ;
- de couvrir autant d'aspects des systèmes multi-agents que possible incluant modèle agent, interaction, coordination, organisation, etc ;
- d'être simples à configurer et à évaluer. Cela se traduit par une bonne documentation, une disponibilité de téléchargement, une procédure d'installation

simple et un support de multiples plates-formes.

Pour être encore plus certains d'effectuer les bons choix, nous éviterons délibérément les plates-formes qui :
- sont toujours à un stade expérimental, abandonnées, ou distribuées confidentiellement ;
- reposent sur des modèles vagues ou non-existants ;
- couvrent seulement un aspect des systèmes multi-agents ;
- sont trop « courtes » au niveau de certaines étapes de la construction du SMA.

Sur base de ces critères, la comparaison mettra en jeu les quatre plates-formes suivantes : **AgentBuilder**, **Jack**, **MadKit** et **Zeus**. Ayant pour objectif d'être le plus complet possible, nous avons ajouté une description et quelques commentaires à propos de l'environnement de développement **Jade** qui n'a sans aucun doute rien à envier au quatuor et qui aurait pu sans conteste y trouver sa place.

Décrivons rapidement par quelques commentaires chacune des plates-formes de développement afin de pouvoir les comparer mieux encore :

AgentBuilder®
AgentBuilder[2] est une suite d'outils intégrés de construction d'agents software intelligents. Il s'agit d'un produit commercial (une version d'évaluation gratuite est disponible) développé par Reticular Systems Inc. et fondé sur les modèles BDI *Agent0* [Sho91] et *Placa* [Tho93]. Cet outil est remarquable par la haute qualité de ses software et du modèle théorique sous-jacent utilisé.

Jack™
Jack[3] est décrit comme un environnement pour la construction, l'exécution et l'intégration de systèmes multi-agents commerciaux basés sur JAVA et utilisant une approche basée sur les composants. Il est développé par Agent Oriented Software Pty. Ltd. (une version est gratuite sous une licence d'utilisation de 60 jours) et basé sur le modèle BDI dMARS [dKLW98b] développé par l'*« Australian Artificial Intelligence Institute »*. Jack se concentre principalement sur l'étape de développement. Les étapes d'analyse et de design sont uniquement mentionnées dans [BRHL99]. Le *toolkit* inclut JDE (Jack Development Environment), un outil graphique pour gérer les projets, le compilateur JAL (Jack Agent Language), qui traduit les programmes JAL en programmes JAVA purs, et une librairie de classes, appelée « Jack Agent Kernel ».

MadKit
MadKit[4] est une plate-forme multi-agents basée sur JAVA et construite sur un modèle organisationnel. Elle est développée par Olivier Gutknecht et Jacques

[2] `http://www.agentbuilder.com/`.
[3] `http://www.agent-software.com.au/`
[4] `http://www.madkit.org/`

Ferber au LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier), un laboratoire de recherche publique en France. La plate-forme est gratuite en cas d'utilisation pédagogique. Contrairement aux trois autres plates-formes présentées, MadKit est essentiellement un moteur d'exécution de SMA. Le modèle organisationnel sous-jacent est appelé Aalaadin [FG98] ou AGR (Agent/Groupe/Rôle).

Zeus

Zeus[5] est un environnement intégré pour la construction rapide d'applications basées sur des agents collaboratifs. Il est développé dans le cadre du programme « *Agent Research Programme* » du laboratoire britannique « *British Telecom Intelligent System Research laboratory* ». C'est un software *open source*, téléchargeable sur le site de Zeus. La documentation de Zeus est abondante, et met très fort l'accent sur l'importance de l'aspect méthodologique de Zeus (« *The agent creation methodology is vital to the use of the Zeus toolkit* » [CN99]). La méthodologie Zeus utilise la décomposition en quatre étapes pour le développement d'agents.

Jade

Jade[6] (Java Agent DEvelopment Framework) est une plate-forme gratuite, distribuée par TILAB et entièrement implémentée en JAVA. Elle simplifie l'implémentation d'un système multi-agents via un middleware qui respecte les spécifications FIPA[7] (Foundation for Intelligent Physical Agents) et des outils graphiques qui supportent les phases de débogage et de déploiement. Aucune méthodologie n'est spécifiée pour le développement. Jade fournit des classes qui implémentent « JESS »[8] pour la définition du comportement des agents. Jess est un moteur de règles petit et léger et un des plus rapides disponibles. Il est entièrement écrit en Java par Ernest Friedman-Hill au laboratoire « Sandia National »[9] à Livermore en Californie.

L'outil possède trois modules principaux (nécessaires aux normes FIPA). Le DF *« Director Facilitor »* fournit un service de pages jaunes à la plate-forme. Le ACC *« Agent Communication Channel »* gère la communication entre les agents. Finalement, l'AMS *« Agent Management System »* supervise l'enregistrement des agents, leur authentification, leur accès et utilisation du système. Les agents communiquent par le langage FIPA ACL *« Agent Communication Language »*.

Comme nous l'avons déjà mentionné, cette plate-forme de développement ne fait malheureusement pas partie de la comparaison sur laquelle nous nous basons par la suite pour réaliser notre choix. C'est pourquoi nous la décrivons volontairement plus en détail dès à présent puisque nous désirons envisager toutes les possibilités potentiellement intéressantes à la réalisation de notre solution. Jade est fourni avec un éditeur disponible pour l'enregistrement et la gestion des agents.

[5] `http://www.labs.bt.com/projects/agents/zeus/`
[6] `http://jade.tilab.com/`
[7] `http://www.fipa.org/about/index.html`
[8] `http://herzberg.ca.sandia.gov/jess/`
[9] `http://www.sandia.gov/`

Aucune autre interface n'est disponible pour le développement ou l'implémentation. A cause de cette lacune, l'**implémentation demande beaucoup d'efforts**. Elle nécessite une bonne connaissance des classes et des différents services offerts [GD02].

Par contre, si l'on en croit une étude menée par Pavel Vrba et dont les résultats apparaissent dans son article « *JAVA-Based Agent Platform Evaluation* » [Vrb03], Jade semble être l'outil de développement open-source le plus approprié pour des solutions multi-agents manufacturières et ce sur base de critères beaucoup plus techniques : Jade offre une vitesse d'envoi de message deux fois plus rapide que tous ses autres concurrents et une stabilité d'environnement spécialement lorsque le nombre d'agents déployés est important. Finalement, il semblerait que l'utilisation de la mémoire soit optimisée.

5.3.2 Comparaison des outils

Comme nous l'avons vu en détail au point 3.2 (page 34), quatre étapes sont nécessaires à la construction d'un système multi-agents. Pour rappel, il s'agit de l'*analyse*, du *design*, du *développement* et du *déploiement*. Etant donné l'utilisation de la méthodologie Prometheus afin de réaliser une analyse et un design détaillés, structurés et bien documentés, il est important de signaler que nous nous intéresserons tout particulièrement aux deux dernières étapes, le développement et le déploiement, l'accent étant mis principalement sur la première.

Dans cette optique, il sera très intéressant de vérifier la **complémentarité** de notre plate-forme de développement avec la méthodologie employée au préalable et qui satisfait aux deux premières phases de la construction du SMA.

Venons-en donc concrètement à la comparaison des différents outils, en commentant chacun d'entre eux et en mettant en avant leurs forces et leurs faiblesses tout en gardant à l'esprit les qualités requises par chaque étape, qualités que nous avons étudiées au point 3.4 (page 35).

AgentBuilder®

Développement

La phase de développement consiste à définir le comportement des agents. C'est aussi durant cette étape que les actions externes et les interfaces graphiques de l'utilisateur sont intégrées dans l'agent. Les outils graphiques supportent toutes ces tâches. Cette étape est uniquement applicable à des agents qui utilisent l'architecture BDI. Les outils graphiques simplifient la construction des règles de comportement des agents, évitant ainsi des erreurs syntaxiques et sémantiques durant leur construction.

Commentaire

La documentation de la plate-forme couvre presque toutes les étapes, de l'analyse au développement ce qui est un bon point. Les outils software sont un autre

point positif. Ils couvrent eux aussi presque tous les différents aspects des étapes et établissent des liens entre eux. Le désavantage d'une telle plate-forme est que la versatilité de l'outil est limitée. Les systèmes multi-agents construits avec AgentBuilder sont composés de façon homogène d'agents qui utilisent le modèle d'agents AgentBuilder. Il n'est pas facile d'intégrer des agents qui utilisent d'autres modèles. Comme tout outil complexe, AgentBuilder est long et difficile à apprendre, mais une fois que l'environnement est maîtrisé, cela devient très productif.

Jack™

Développement

Le développement est une combinaison de code JAVA normal pour les classes élémentaires et de JAVA étendu -JAL pour *Jack Agent Language*- pour les composants spécifiques aux agents. Les extensions permettent de décrire les comportements, capacités, plans, événements et bases de données relationnelles dans le code JAVA. Le compilateur de Jack traduit ensuite ce JAVA étendu en JAVA pur. Un outil graphique est également fourni pour faciliter le management de larges projets. L'étape de développement de Jack est applicable aux systèmes multi-agents composés d'agents BDI éventuellement interfacés avec d'anciens systèmes. Jack n'impose pas d'utiliser son propre modèle BDI, les autres modèles d'agents pouvant être utilisés (mais doivent être implémentés), tout en continuant à bénéficier des services du Kernel de Jack, le *« Jack Agent Kernel »*.

Commentaire

Jack sort du rang par le haut degré d'orientation agent de sa programmation. Cela mène à une haute versatilité : l'architecture des agents peut s'étendre de simples comportements réactifs codés en Java à du BDI intégral en utilisant l'architecture fournie ou une autre. Petit bémol, la documentation fournie est très technique, et ne couvre pas les aspects méthodologiques, spécialement pour les étapes d'analyse et de design. Le déploiement manque également de support.

MadKit

Développement

Cette étape inclut le choix du modèle d'agents, son implémentation, et l'implémentation des stratégies de protocole d'interaction. Aucun modèle d'agent n'est fourni, il doit être implémenté en Java « *from scratch* », ou modifié (en partant d'un modèle existant) pour fonctionner avec MadKit. Puisque aucune hypothèse n'est faite sur le modèle d'agent, tout type de modèle peut être utilisé (les modèles simples sont à préférer). Mais comme tout le code de l'agent doit être implémenté en Java, le développement des agents cognitifs complexes peut être une tâche difficile et fastidieuse. Par chance, les modèles d'agent implémentés pour MadKit peuvent être réutilisés au fil des projets.

Commentaire

La caractéristique principale de MadKit est d'être un moteur d'exécution multi-agents. Le développement et le déploiement sont donc facilités puisque la plate-forme se concentre sur l'infrastructure des agents. Le manque de méthodologie (à l'exception du modèle Aalaadin, qui est plus un modèle organisationnel descriptif qu'une méthode de conception) est en voie d'être surmonté même si quelques travaux sont encore nécessaires. Le principal défaut de MadKit est que la construction des agents impose une lourde tâche de codage de la part du développeur puisqu'il n'existe pas de modèle au préalable. D'un point de vue plus positif, cela ajoute de la flexibilité et tout programmeur peut commencer par écrire son propre agent, même s'il ne possède pas un important « *background* » des systèmes agents, rendant ainsi MadKit très intéressant dans un contexte didactique.

Zeus

Développement

Cette étape est documentée au moyen d'études de cas et de « *l'Application Realisation Guide* ». Les cinq activités impliquées dans le processus de développement d'agents sont toutes supportées par des outils graphiques. Ces cinq activités sont : *Création de l'Ontologie*, *Création de l'Agent*, *Création des Utilités de l'Agent*, *Configuration des Tâches de l'Agent* et *Implémentation de l'agent*. Ces activités nécessitent une bonne connaissance des outils qui sont heureusement fort bien documentés. Cette étape n'est applicable qu'à des modèles d'agent Zeus. Les ontologies peuvent être facilement réutilisées. Il n'y a pas encore de support pour la réutilisation d'agents entre projets. La modularité générale de l'outil de développement permet la réutilisation de fonctionnalités d'agents utilisées fréquemment mais cela nécessite de retourner dans le code Java.

Commentaire

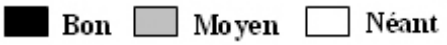
La principale particularité de Zeus est son intégration de toutes les étapes, du design au déploiement. La plate-forme fournit des outils théoriques et pratiques et utilise des techniques actuelles de « *software engineering* » (« *design patterns* », « *UML* »). Ses documents méthodologiques se concentrent sur le *comment faire* et pas uniquement sur le *que faire*. Cependant, même si Zeus est modulaire, un seul modèle d'agent est supporté, ce qui limite considérablement l'éventail possible de systèmes multi-agents. De plus, comme tout outil complexe, Zeus est long et difficile à maîtriser, mais le bénéfice de productivité est important une fois l'outil maîtrisé.

Résultats de la comparaison

La figure 5.1 ci-dessous illustre les valeurs qualitatives des différents critères (Complétude, Applicabilité, Complexité et Réutilisabilité) que nous avons étudiés au point 3.4 (page 35). Comme nous l'avons déjà mentionné auparavant, à ce point

du travail (nous disposons d'une méthodologie et nous cherchons une plate-forme satisfaisante pour l'étape de développement), nous accordons une importance primordiale à la couverture de la phase de développement par la plate-forme. Dans ce contexte, il n'est pas difficile d'avancer que **Jack a et aura notre faveur comme environnement de développement de notre application**. De plus, ses avantages ne s'arrêtent pas uniquement à cela, c'est pourquoi nous justifions notre choix au point 5.3.3 (page 62).

Légende :



Phase	Analyse				Design				Développement				Déploiement			
Qualité	C	A	C	R	C	A	C	R	C	A	C	R	C	A	C	R
AgentBuilder	■	■	■	■	■	■	▒	▒	■	▒	▒	■	■	▒	■	
Jack					▒	▒	▒		■	■	■	■				
MadKit					▒	■	■	■	▒	▒	▒	■	■	■	■	■
Zeus	■	■	■	■	■	■	■	■	■	▒	▒	▒	■	■	■	

TAB. 5.1 – Valeurs qualitatives des différents critères (Complétude, Applicabilité, Complexité et Réutilisabilité) utilisés pour analyser les plates-formes [DW03].

Discussion

D'un point de vue méthodologique, nous pouvons observer qu'il existe de grandes différences entre les plates-formes, de la méthodologie fondamentale dans Zeus à la quasi absence d'outils méthodologiques dans Jack ou MadKit. Le point de départ d'une plate-forme est souvent un outil d'implémentation mais cet outil n'est pas suffisant. Le développeur a besoin d'un manuel pour l'utiliser et donc aussi d'une procédure à suivre afin de pouvoir, au moyen de tâches simples, mener le projet et l'application du stade d'analyse au stade du déploiement.

D'un point de vue technique, AgentBuilder et Zeus sont davantage des « éditeurs BDI » alors que Jack est un langage de programmation orienté agents. Les premiers sont plus efficaces parce qu'ils permettent de programmer avec un haut niveau d'abstraction mais ils sont dévoués à une architecture agent unique. A l'opposé, les langages de programmation agent sont plus versatiles, aux frais d'une écriture de code plus lourde et fastidieuse due à un niveau d'abstraction plus bas.

L'idéal serait une plate-forme modulaire, accompagnée d'une riche bibliothèque de modèles agent et où chaque modèle possèderait son propre éditeur. Cette plate-forme combinerait polyvalence et programmation de haut niveau. Malheureusement, cette plate-forme parfaite n'existe pas encore même si des travaux actuels vont dans cette direction. C'est le cas de la plate-forme MASK [OBDK00], toujours en cours de développement, qui contient cette librairie de modèles agents

et est accompagnée de la méthodologie « Vowels » [DOBR00] qui devrait couvrir au mieux possible les quatre étapes de construction d'un logiciel et qui propose une librairie de modèles agents.

5.3.3 Justification de notre choix

Comme nous l'avons déjà annoncé, nous emploierons Jack comme plate-frome de développement pour notre application et ce pour les diverses raisons suivantes :

- l'approche de Jack est très **modulaire** : la plupart des composants du *framework* peuvent être mis au point, améliorés, façonnés et adaptés ;
- elle est très **complémentaire** avec la méthodologie d'analyse Prometheus. L'environnement de développement de Jack ou JDE pour « *Jack Environment Development* » supporte la méthodologie Prometheus en ce sens que les concepts utilisés lors de la conception du système sous Jack correspondent aux concepts des derniers « *artifacts* » développés dans Prometheus lors de la phase du design detaillé.
- au point 5.3.2 (page 58), nous avons déclaré : « *la documentation de Jack est très technique, et ne couvre pas les aspects méthodologiques spécialement pour les étapes d'analyse et de design* ». Dans notre cas, cela ne nous pose en aucun cas de problème puisque d'une part la plate-forme et plus particulièrement ses concepts sont complémentaires avec les concepts sous-jacents de notre méthodologie et que d'autre part, les phases d'analyse et de design sont prises en compte entièrement par Prometheus ;
- après analyse de la figure 5.1, la plate-forme s'avère être la meilleure au niveau de la couverture de la phase de **développement** ;
- toujours d'un point de vue plus technique, les manuels fournis sont très complets et aident considérablement tout développeur lors de ses premiers pas ;
- la plate-forme ne dispose pas de support spécifique au déploiement mais le lancement des classes correspondantes aux agents dans des terminaux, séparés de préférence, est très simple ;
- à l'instar de l'outil Jade, Jack répond totalement aux normes FIPA. Pour rappel, les standards FIPA ont été reconnus comme étant une propriété cruciale des plates-formes de développement assurant notamment l'interopérabilité entre différents types d'agents. En effet, la plate-forme peut interopérer avec d'autres plates-formes telles que Jade grâce au protocole de transport de message FIPA basé sur HTTP ;
- la plate-forme Jack est répandus et mise à jour régulièrement ;
- elle est entièrement écrite en Java, dont les qualités ne sont plus un secret : flexibilité, extensibilité, familiarité, portabilité et interopérabilité ;
- Jack est un produit développé en Australie, terre d'accueil de notre stage ;
- l'UTS, *University of Technology of Sydney*, entretient une relation particulière avec AOS, la compagnie qui développe le produit. Elle dispose des dernières mises à jour de la plate-forme et plusieurs chercheurs sont en contact régulier avec les concepteurs de la plate-forme. Utiliser l'environnement de développement Jack nous permet donc de bénéficier d'un support auprès de

divers chercheurs travaillant au sein de l'équipe qui nous accueille.

5.4 JACK Intelligent Agents

5.4.1 Introduction

Notre choix effectué, nous allons terminer ce chapitre en décrivant un peu plus en détail la plate-forme de développement Jack [HRHL01, Sof04] en termes d'objectifs, de composants, architectures, etc. Agent Oriented Software Pty. Ltd. (AOS), basé à Melbourne, Australie, a construit JACK Intelligent Agents^{TM}, un *framework* en Java, pour le développement de systèmes multi-agents. L'objectif de la compagnie est de fournir une plate-forme commerciale, industrielle et de recherche. A cette fin, le *framework* fournit une **implémentation haute performance et légère de l'architecture BDI** et peut être facilement étendu pour supporter différents modèles d'agents ou des besoins spécifiques aux applications.

A l'origine, Jack fut développée pour satisfaire aux objectifs suivants :

- fournir aux développeurs un produit robuste, stable et léger ;
- satisfaire à une variété de besoins des applications pratiques ;
- faciliter le transfert de technologie de la recherche à l'industrie ; et enfin,
- permettre davantage de recherches expérimentales.

5.4.2 Qu'est-ce qu'un agent JACK ?

Les agents utilisés dans Jack sont des *agents intelligents*. Ils modèlisent leur comportement, leur raisonnement en suivant le modèle théorique *Belief Desire Intention* que nous avons déjà évoqué.

Suivant ce modèle, les agents intelligents JACK sont des composants software autonomes qui ont des objectifs explicites à atteindre et des événements à traiter. Un agent JACK est un composant software qui peut faire preuve de comportement de raisonnement sous des stimuli, soit proactif (orienté objectifs) soit réactif (orienté événements). Pour rappel, chaque agent possède :

- un ensemble de **croyances** ;
- un ensemble d'**événements** ;
- un ensemble d'**objectifs** (soit à la demande d'un agent externe, suite à un événement soit quand une ou plusieurs de ses croyances changent) ;
- un ensemble de **plans**.

5.4.3 Les composants de JACK

JACK Intelligent Agents^{TM} est distribué sous la forme de modules dont nous reprenons les plus importants :

- **JACK Agent Language :** Ce langage est le langage de programmation utilisé pour décrire un système orienté agents. JAL est un super ensemble de JAVA : il inclut complètement la syntaxe JAVA et l'étend avec des constructions qui représentent les caractéristiques orientées agents. Comme c'était le cas pour le langage C++ vis-à-vis du langage C, JAL fait plus qu'étendre

les fonctionnalités de JAVA, il fournit aussi un *framework* pour supporter entièrement un nouveau paradigme de programmation.

Le JACK Agent Language étend JAVA pour supporter la programmation orientée agents de la façon suivante :
- il définit des nouvelles classes, interfaces et méthodes de base ;
- il fournit des extensions syntaxiques à JAVA pour supporter les nouvelles classes, définitions et commandes orientées agents ;
- il fournit des extensions sémantiques à JAVA (ce qui signifie des exécutions différentes en fonction, par exemple, des types d'objet) pour supporter le modèle d'exécution requis par un système software orienté agents.

Toutes les extensions du langage sont fournies en tant que plug-in. Cela rend le langage aussi extensible et flexible que possible. Cette flexibilité est importante car elle facilite les recherches en cours dans le domaine de la programmation orientée agents. Chacune des extensions de JAL est strictement « *typée* ». Ceci réduit le « *casting* » de type implicite et la possibilité de faire des erreurs pour le développeur. Le typage strict permet aussi une compilation plus efficace des programmes par le JACK Agent Compiler.

L'implémentation du système et plus précisemment l'écriture du code source peut se faire soit par l'intermédiaire de l'environnement de développement Jack soit au moyen d'un éditeur classique de notre choix. Lors du développement de toute application Jack, des codes sources vont être créés pour certaines ou toutes les entités suivantes :
- *événement(s)* JACK ;
- *plan(s)* JACK ;
- *agent(s)* JACK ;
- *capacité(s)* JACK ;
- *vue(s)* JACK ;
- *ensemble de croyances* JACK.

Plus concrètement, chacun des fichiers représentant ces entités doit avoir le même nom de base que le nom de l'entité définie dans le fichier et il est caractérisé par une extension qui est fonction du type de l'entité Jack contenue dans ce fichier. La table 5.2 reprend l'ensemble des nouvelles extensions de fichier et l'entité Jack pour laquelle chacune est utilisée.

- **JACK Runtime Environment :** Le « JACK Agent Kernel » est un moteur d'exécution pour les programmes écrits en JACK Agent Language. Il fournit un ensemble de classes qui donnent aux programmes écrits en JAL leur fonctionnalité orientée agent. La plupart des classes s'exécutent en arrière-plan et implémentent l'infrastructure et la fonctionnalité sous-jacentes que les agents requièrent, pendant que les autres sont utilisées explicitement dans les programmes écrits en JAL puisque ces derniers en héritent et contiennent des appels aux premières comme cela est requis pour

Extension	Entité JACK
`.jack`	Toute définition d'objet JACK
`.agent`	Définition d'agent JACK
`.cap`	Définition de capacité JACK
`.plan`	Définition de plan JACK
`.event`	Définition d'événement JACK
`.bel`	Définition d'ensemble de croyances JACK
`.view`	Définition de vue JACK

TAB. 5.2 – Entités JACK et extensions de fichier correspondantes

fournir leur(s) fonctionnalité(s) aux agents.
Le support lors de l'exécution des agents inclut :
- la gestion automatique de la concurrence entre les tâches poursuivies en parallèle (*Intentions dans la terminologie BDI et donc concrètement l'exécution d'un ou de plusieurs plans*) ;
- le comportement par défaut d'un agent en réaction à des événements, des échecs d'actions ou de tâches, etc ;
- une infrastructure de communication légère et hautement performante pour des applications multi-agents.

Il est important de signaler que le Kernel de Jack supporte l'exécution d'agents multiples dans un processus unique, d'agents multiples dans plusieurs processus et toute combinaison des deux.

- **JACK Compiler :** Le compilateur convertit les additions syntaxiques (écrites en code JAL) en des commandes et classes en JAVA pur qui peuvent être appelées par d'autres codes écrits originellement en JAVA. Le compilateur transforme également partiellement le code des plans afin d'obtenir des sémantiques correctes correspondant à une architecture BDI. Finalement, il appelle le compilateur JAVA pour générer des éxécutables.

- **Agent Development Environment :** L'environnement de développement est une interface graphique pour la consultation et la manipulation des applications Jack.

5.4.4 Conclusion

Comme nous avons pu nous en apercevoir auparavant, JACK est un environnement de développement d'applications multi-agents modulaire, léger et performant. Même si l'adoption de JAVA signifie la perte d'une certaine puissance que peuvent offrir les *frameworks* basés sur des langages fonctionnels ou logiques, cette perte est compensée d'une part par sa disponibilité universelle (bien résumée dans le fameux slogan de Sun Microsystems *« Compile Once, run everywhere »*) et d'autre part, surtout par l'approche modulaire de JACK. De plus, sur le plan de la vitesse d'exécution d'un agent écrit en JACK, les performances affichées par JACK sont tout à fait comparables aux implémentations en C ou C++.

C'est avec des concepts acquis et des choix effectués tant au niveau de la méthodologie AOSE que de l'outil de développement que nous pouvons poursuivre notre travail et confronter toutes les notions et concepts sous-jacents du paradigme multi-agents au domaine du workflow management en prenant soin, dans le chapitre suivant, de présenter ce domaine de la façon la plus précise et concise possible.

Troisième partie

Systèmes Multi-Agents et Workflow Management

Chapitre 6

Introduction au Workflow Management

Sommaire

6.1 Introduction

Au cours des deux premières parties de ce travail, nous nous sommes attachés d'une part à définir les notions importantes relatives aux systèmes multi-agents et aux modèles adjacents comme le modèle d'attitudes mentales *Belief-Desire-Intention* (chapitres 1 et 2), et d'autre part à présenter les méthodologies et plate-formes de développement actuelles (chapitres 4 et 5) et y justifier nos choix.

Ces différents concepts fixés, nous allons à travers cette troisième partie nous rapprocher plus encore du sujet de ce mémoire et répondre à l'un des objectifs

principaux qui est d'étudier le couple système multi-agents & workflow management[1] sous différents angles (efficacité, avantages, opportunités, etc). Dans la suite, et pour plus de facilité, nous utiliserons parfois les abréviations respectives *SMA* et *WM* pour ces deux concepts.

Avant d'en arriver là, et puisque nous avons déjà longuement parlé des SMA, nous voudrions en guise de chapitre introductif de cette troisième partie, nous attarder sur le processus d'organisation optimale des activités qu'est le workflow management. D'une étude historique aux caractéristiques essentielles du WM, nous aborderons à travers ce chapitre 6 les différents points permettant de répondre de la meilleure façon qui soit à la question : **Qu'est-ce que le workflow management** ?

6.2 Définitions

Nombreuses sont les ouvrages abordant le sujet du workflow management. Il n'est pas simple (et peut-être même impossible) de trouver une définition qui fasse figure de référence dans les milieux concernés. La mixité des secteurs dans lesquels le workflow management peut trouver sa place (association, université, entreprise commerciale, industrie,etc) mais aussi la variation de l'influence et de l'impact qu'il peut avoir sur l'organisation ont comme conséquence l'existence d'une multitude de définitions qui, selon la source et l'environnement auquel le WM est appliqué, varient et insistent différemment sur certains objectifs, participants ou procédés.

Les définitions suivantes proviennent de trois domaines d'application tout à fait différents : la première définition provient du domaine de la consultance, la deuxième du domaine de la recherche et la troisième du secteur industriel.

Hales et Lavery proposent une définition très complète et adéquate du workflow management [HL91] :

> Un software de workflow management est un système logiciel, proactif, qui gère le flux de travail entre **les participants**, selon une procédure définie. Cette dernière consiste en un certain **nombre de tâches**. Le système coordonne les participants, systèmes et utilisateurs, ensemble avec **les ressources de données** appropriées, pour atteindre des objectifs définis pour des deadlines[2] déterminées. La coordination implique de transférer des tâches de participant en participant dans une séquence correcte, en s'assurant que chacun remplit son contrat, en prenant des actions définies par défaut lorsque cela est nécessaire.

[1] Termes couramment utilisés dans les milieux francophones, nous les utiliserons également dans la langue de Shakespeare au cours de notre rédaction. La traduction de ces termes en français est en effet beaucoup moins parlante.

[2] Terme anglais souvent utilisé dans le cadre de projet informatique, synonyme de *date limite*.

La définition ci-dessus caractérise bien les composants fondamentaux d'un workflow que nous répétons à nouveau : des **tâches** ordonnées, des **participants** qui doivent exécuter les tâches et des **ressources de données** nécessaires pour accomplir ces tâches. De plus, cette définition caractérise très bien les services rendus par le système de WM : *transférer les tâches* de participant en participant, *contrôler* que les participants apportent bien leur(s) contribution(s) et enfin, *fournir certains traitements* d'exception. Une autre définition est donnée par Reinwald[Rei94] :

> Un système de WM est un système actif qui gère le flux des processus de business réalisés par plusieurs personnes. Il attribue la bonne donnée à la bonne personne avec les bons outils au bon moment.

Cette définition est un peu réductionniste dans le sens où elle ne fait pas de distinction entre des processus de business et le workflow management, qu'elle passe sous silence les participants « *systèmes* », et n'attache de l'importance qu'aux personnes. Par contre, elle mentionne bien trois composants importants supplémentaires : les **données**, les **outils** et le **bon moment d'exécution**. Nous ajoutons une troisième définition qui nous vient du secteur industriel et plus spécifiquement du consortium industriel appelé *Workflow Management Coalition*[3] [Coa95] :

> Un système de WM est un système qui fournit de l'automatisation procédurale de processus de business par la gestion de la séquence des activités et l'invocation de ressources appropriées humaines et/ou IT associées en fonction des étapes.

Chacune de ces définitions apporte une brique à l'édifice et nous permet de mieux comprendre et appréhender ce qu'est le workflow management.

6.3 Notes historiques

6.3.1 Deux conditions préalables

De nombreux analystes de l'industrie IT affirment que le workflow management a été la technologie software des années 90 [JB96]. Pourtant, depuis bien plus longtemps, les travailleurs tentent d'intégrer les technologies de l'information comme support à leur travail. Avant d'étudier historiquement les origines du WM, il est intéressant de se poser la question suivante : pourquoi le WM n'est-il pas aussi la technologie software des années 80 ? Quel(s) changement(s) entre les années 1980 et les années 1990 ont pu rendre le WM aussi populaire ?

Dès les années 1970, le travail de bureau était fortement influencé par les technologies informatiques, notamment par les systèmes de bases de données qui

[3] Il s'agit de l'organisation internationale composée des vendeurs, utilisateurs, analystes et groupes de recherche ou universitaires pour la définition des standards du workflow. Les dernières évolutions et informations, les derniers standards et modèles et toutes les informations liées au workflow sont disponibles sur le site officiel de la WfMC : `http://www.wfmc.org`.

évoluèrent rapidement ainsi que la puissance de calcul et d'intelligence disponible. Cependant, un manque d'intégration des produits disponibles avait pour conséquence de faibles améliorations sur le terrain.

Toujours selon Stefan Jablonski et Christoph Bussler [JB96], deux tendances fondamentales ont contribué au large avènement des technologies du WM : d'un part, **les énormes progrès technologiques** qui caractérisent les années 80 et 90 et d'autre part, **un changement dans le développement des systèmes applicatifs** : l'objectif n'est plus l'automation de morceaux d'application mais bien le développement complet de solutions intégrées.

Les progrès technologiques peuvent être démontrés à travers les « *trois lois* » [GR93] qui décrivent le développement du hardware de façon significative :

- la loi de Moore commente le progrès des mémoires électroniques : d'une moyenne de 1Kb par puce en 1970, la capacité d'une puce mémoire a augmenté d'un facteur de 4 tous les 3 ans ;
- la loi de Hoagland déclare que la densité de lecture et écriture des disques et bandes magnétiques a été multipliée par 10 chaque décennie ;
- la loi de Joy prédisait, entre les années 1984 et 2000, un taux mips[4] des machines Sun MicroSystems à $2^{année-1984}$. Cette formule est sans doute trop optmiste mais la tendance générale peut être reconnue et est impressionante.

Au-delà de ces considérations au niveau du développement de la mémoire et des processeurs, nous pouvons ajouter les développements impressionnants au niveau des réseaux de télécommunications. De plus, tous ces progrès dans les technologies hardware permettent un déploiement des logiciels d'une façon révolutionnaire : pour la première fois, les environnements logiciel intégrés peuvent être établis aussi bien localement que globalement.

Mais comme nous l'avons mentionné auparavant, les progrès technologiques n'ont pas contribué seuls au développement des technologies de WM. Une nouvelle approche de définition et de développement des systèmes a également participé au développement des technologies du WM. D'une vue centrée sur la *tâche* ou la *donnée* nous sommes passés à une vue centrée *processus (de business)* ou *travail* qui caractérise bien les approches modernes du développement d'applications.

6.3.2 Les origines du Workflow Management

Les deux prérequis à l'essor des technologies du workflow management étant acquis, nous allons rapidement dresser un inventaire des développements technologiques qui sont à l'origine, d'une façon ou d'une autre, des technologies du WM. Pas moins de sept technologies software sont considérées comme étant des « ancètres conceptuels » du workflow management, à savoir :

- la bureautique ;
- la gestion de base de données ;

[4] Unité de mesure de la vitesse d'exécution d'un ordinateur (de son processeur). *MIPS* signifie Million d'instructions par seconde.

- l'e-mail ;
- la gestion des documents ;
- la gestion des processus sofware ;
- la modélisation des processus business ; et enfin,
- la modélisation d'entreprise et architecture.

La bureautique peut être considérée comme la principale origine du workflow management [EN80] à travers le contrôle et la conduite de l'exécution des processus de business. Nous laissons au lecteur le soin de consulter l'annexe C pour mieux évaluer les apports de chacune de ces sept technologies au niveau du workflow management.

6.4 Domaine d'application

Les « *Computer Supported Cooperative Work* » (CSCW) et/ou « *groupware* » sont considérés comme un domaine d'application possible et approprié pour le workflow management [MB92].

6.4.1 Computer Supported Cooperative Work et Groupware

Les définitions suivantes permettront au lecteur de mieux cerner la différence ou plutôt la relation existant entre ces deux concepts. Wilson [Wil91] donne la définition suivante des CSCW :

> ***CSCW*** est un terme générique qui combine la compréhension de la façon dont les personnes travaillent en groupe avec les possibilités des technologies des réseaux d'ordinateurs, du *hardware*, *software*, des services et des techniques associées.

Ellis *et al.* [EGR91] caractérisent les CSCW comme suit :

> Comptant sur l'expertise et la collaboration de nombreux spécialistes en informatique mais aussi de sociologues, les ***CSCW*** s'attachent à regarder comment les groupes travaillent et à chercher à découvrir comment les technologies (spécialement les ordinateurs) peuvent les aider à travailler.

Ces deux définitions mettent l'accent sur les aspects conceptuels. Elles insistent plus sur le fait que les technologies de l'ordinateur *doivent être déployées* pour les CSCW que sur *la façon* dont ces technologies devraient être déployées.

Au contraire, dans les définitions de la notion de *groupware*, les aspects software sont les centres d'intérêt. Johansen [Joh88] définit le *groupware* comme un terme représentant des assistances spécialisées par ordinateur ou *« Specialized computer aids »* :

> ***Groupware*** est un terme générique pour l'assistance par ordinateur qui est utilisée par des groupes de travail collaboratifs. Typiquement,

ces groupes sont des petites équipes orientées projets qui ont des tâches importantes et des *deadlines* serrées. *Groupware* peut impliquer du support au niveau du software, hardware, services et/ou processus de groupe.

En résumé, le terme « CSCW » est souvent utilisé lorsque l'on parle de problèmes théoriques ou conceptuels, et nous utiliserons le terme « groupware » pour faire référence à des problèmes systèmes.

6.4.2 Analyse du terme « CSCW »

Une approche très intéressante pour analyser le terme « CSCW » est rapportée par [BS95] qui interprète ce terme en inspectant chacune de ses lettres :

- C : une technologie basée sur le ***Computer***, l'ordinateur.
- S : la technologie doit **S**upporter certains types d'application.
- C : ces types d'application mettent l'accent sur la **C**oopération.
- W : cette coopération, supportée par l'ordinateur, produira certains travaux ou ***Work***.

Assez intelligement, les auteurs pratiquèrent la même analyse, avec le même mot, mais dans l'autre sens :

- W : le travail ou ***Work*** à accomplir est mis en évidence.
- C : l'accomplissement de ce travail est basé sur la division du travail et la **C**oopération des travailleurs.
- S : le succès du travail depend de l'adéquation du **S**upport.
- C : l'ordinateur ou **C**omputer est une assistance qui peut supporter le travail coopératif.

En résumé, nous remarquons l'importance que revêtent les aspects ***ordinateur*** et ***travail*** par rapport à la notion de CSCW et auxquels l'***environnement organisationnel*** (ou infrastructure) peut sans conteste être ajouté. Le CSCW traite donc des inter-relations et des interdépendances entre les trois thèmes travail, technologie (ordinateur) et organisation que nous retrouvons dans la figure 6.1. En conséquence, ces trois notions sont également le centre d'intérêt du workflow management.

Au coeur du système se trouve la notion de **processus de groupe** qui sert à lier les trois éléments travail, organisation et technologie. Cet élément est caractérisé par un aspect statique et un aspect dynamique. L'*aspect statique* comprend les facteurs suivants :

- les objectifs du processus de groupe ;
- les structures organisationnelles c'est-à-dire les personnes qui y participent ;
- un protocole de groupe qui l'orchestre.

Quant à l'*aspect dynamique*, il est défini par les éléments suivants :

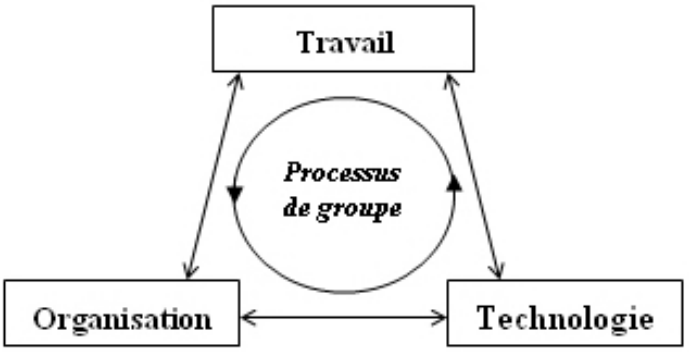


FIG. 6.1 – Les trois thèmes constitutifs du CSCW [JB96].

- une infrastructure IT (hardware et software) et non-IT (bâtiments et installations) ;
- des documents qui décrivent le processus de groupe ;
- des activités de groupe réalisées pendant le processus ;
- des sessions de groupe qui se composent d'activités auxquelles les personnes sont assignées ;
- un état du groupe qui reflète l'état courant du processus.

En conclusion, comme nous avons pu aisément le constater, la notion de processus de groupe est très importante dans les systèmes CSCW. C'est notamment sur base de celle-ci qu'une classification des systèmes CSCW s'opère et plus précisément sur base des activités et des sessions de groupe ou plutôt la façon dont ceux-ci sont implémentés. Le prochain point détaillera cette observation.

6.4.3 Classification

En plus de caractériser les CSCW, Ellis *et al.* [EGR91] introduisent une classification classique pour les systèmes *groupware* (et donc également pour les applications CSCW). Cette classification est appelée matrice espace temps (table 6.1) et classifie les *groupware* sur base de deux critères :

– le **temps** où les activités de groupe ont lieu et,
– le **lieu** où les sessions de groupe prennent place.

	Même moment	*Différents moments*
Même endroit	interaction face-à-face	interaction asynchrone
Différents endroits	interaction synchrone distribuée	interaction asynchrone distribuée

TAB. 6.1 – Matrice Espace Temps des applications Groupware [JB96].

L'exemple le plus classique de face-à-face est une réunion conventionnelle : les personnes sont assises ensemble au même moment au même endroit et participent à une activité de groupe.

Les systèmes de conférence assistés par ordinateur [GS86] permettent à des personnes localisées à différents endroits de tenir des réunions (synchrones) « distribuées ».

Des éditeurs multi-utilisateurs tels que ForComment [Opp88] qui permet l'édition asynchrone de documents sont un bon exemple d'interaction asynchrone (même endroit, moment différent).

Enfin, les **interactions distribuées asynchrones** sont une forme spéciale d'applications CSCW/Groupware. Les personnes coopèrent à différents moments à partir de différents endroits. Le workflow management supporte très bien ce type de coopération.

6.5 Attentes et Craintes

6.5.1 Attentes

La contribution du workflow management à l'efficacité managériale et organisationnelle se concrétise sous la forme des implications positives au business suivantes [JAD+] :

- ***Augmentation de la qualité de service***. La provision de services est améliorée au niveau de l'efficacité et de l'efficience. Les délais non-nécessaires sont éliminés grâce au routage automatisé et à la division du travail, le taux d'échec est réduit en raison de vérifications de validité et la conformité aux règles fédérales des états ou des entreprises est assurée.
- ***Amélioration des services aux clients***. Deux services majeurs peuvent être fournis aux clients : un service d'information instantané en réponse aux requêtes et une flexibilité qui permet de démarrer des activités alternatives pour offrir de nouveaux services/produits.
- ***Augmentation de la productivité***. Suite à l'élimination des délais et un « débit » de travail plus élevé, la productivité peut être augmentée.
- ***Réduction des coûts***. En raison d'un « débit » de travail plus élevé, le temps nécessaire passé sur un service/produit particulier diminue.
- ***Réduction de la vulnérabilité***. L'introduction des technologies du WM nécessite l'investigation autour des processus de groupe (le coeur du workflow management) qui devront être implémentés. L'augmentation de la connaissance résultant de cette investigation à propos de ces processus de groupe peut être utilisée pour replanifier les travaux, les différer ou leur mettre des priorités pour réduire la vulnérabilité de l'organisation.

6.5.2 Craintes

Néanmoins, comme c'est très souvent le cas lors de l'introduction d'une nouvelle technologie, des craintes et des réserves doivent être exprimées en raison des

conséquences inconnues qui pouvaient apparaître suite au déploiement de cette technologie. C'est d'autant plus vrai dans un contexte où les personnes s'inquiètent de leur avenir au sein de l'organisation. Pour certains, le workflow management peut être considéré comme une camisole de force qui contrôle et supervise les utilisateurs humains qui ne sont plus que des agents stupides qui doivent réagir à tout ce que le système prescrit. Concrètement, les craintes et réserves peuvent être [JAD+] :

- ***Un ontrôle trop rigide***. Le système de WM peut contrôler et diriger les utilisateurs à un degré inacceptable.
- ***Une surveillance trop accrue***. Les personnes craignent la possiblité que les managers puissent inspecter à outrance leur travail. En effet, en principe, tous les types d'analyse sont possibles : quel employé a passé combien de temps sur quelle tâche et quelle fut la qualité de ce travail ?
- ***Trop peu de fonctionnalités***. Le WM ne pourrait à lui seul résoudre tous les problèmes majeurs d'un domaine d'application. En effet, le WM est souvent considéré comme une panacée mais le problème clé est de bien équilibrer son utilisation et de ne pas déployer aveuglément le WM à tous les problèmes.
- ***Peu de flexibilité***. Le workflow management ne peut pas être ajusté facilement à toute nouvelle exigence, demande de la part du client ou de l'environnement du système.

Personne ne peut nier que le WM impose un contrôle important sur les processus (de groupe) et, en conséquence, aussi sur les personnes qui sont impliquées dans la réalisation de ces processus. Cependant, cette observation est commune à la plupart si pas toutes les technologies de l'ordinateur.

Si le workflow management est utilisé modérément, il peut être **utile** et représenter **un guide précieux** d'un bout à l'autre de l'exécution d'un processus (de groupe).

6.6 Les générations de Workflow Management

C'est dans une logique contextuelle et non temporelle que nous terminons ce chapitre introductif au workflow management. Nous introduisons ici la suite de notre exposé et les théories que nous voulons avancer. L'objectif de ce paragraphe est donc très important puisqu'il va nous permettre de mettre en avant l'étroite relation qui existe entre le workflow management et les systèmes multi-agents. Auparavant, comme toute technologie, il sera bon de rappeler à quel stade de développement se situe le workflow management et dans quel contexte cette technologie évolue actuellement.

6.6.1 Sur la route de la maturité

Selon Kenneth R. Abbott et Sunil K. Sarin [AS94], le workflow management évolue en passant par 4 phases :

1. ***Le prototypage***. Des expériences avec des prototypes académiques et commerciaux doivent être réalisées. L'objectif est d'emmagasiner des connaissances au niveau de la modélisation et de l'exécution de WM.
2. ***La conceptualisation***. Sur base de l'expérience acquise lors de la première phase, des modèles, architectures et finalement premières méthodologies sont développés.
3. ***L'outillage***. La prolifération des outils pour le développement d'applications de workflow caractérise cette troisième et avant-dernière phase.
4. ***La standardisation***. Le développement de normes et de standards rend l'utilisation d'une technologie plus efficace et efficiente.

Actuellement, nous pouvons estimer que le WM se situe au niveau de la troisième phase (Figure 6.2). La prolifération de plus en plus importante des outils professionnels nous conforte dans cette idée. Pourtant, selon nous, la phase de conceptualisation n'est peut-être pas totalement terminée. Des modèles et des architectures existent mais rares sont ceux qui font preuve d'une qualité exemplaire. Nul doute que l'arrivée de méthodologies de plus en plus affinées permettra aux outils d'être mieux structurés et finalement au workflow management de poursuivre sa route vers la maturité.

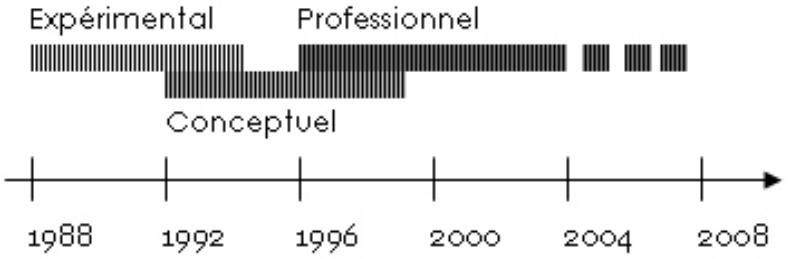


FIG. 6.2 – Phases d'évolution des systèmes de Workflow Management.

L'évolution des technologies du workflow management ne fait pas de doute. Cependant, bien que les systèmes de WM n'en soient plus à leur premier stade, nous sommes relativement loin d'une pleine maturité dans ce domaine. Malgré cela, nous pouvons actuellement nous féliciter de la présence de plus en plus forte de certaines approches très prometteuses et qui font évoluer les systèmes de WM dans la bonne direction.

6.6.2 Vers un Workflow Management basé sur des Sytèmes Multi-Agents

Laissons momentanément de côté le concept de workflow management et concentrons-nous uniquement sur le notion de *workflow*[5]. Le workflow existe depuis la nuit des temps. Seules la prise de conscience de son importance et sa gestion

[5]En francais, flux de travail.

sont beaucoup plus récentes. Organisations, gouvernements et toutes autres entreprises réalisant un « business » ont toujours voulu systématiser leurs activités. Le workflow a donc évolué et l'étude des différentes générations est un outil précieux pour mieux comprendre l'importance actuelle que prend le workflow management dans une entreprise et surtout, la forme qu'il revêt.

Pas moins de cinq générations peuvent être mises en évidence [SH99] :

- ***1ère : Manuel.*** Les bureaucraties font partie de la société humaine et elles impliquent un traitement de l'information. Avant qu'une aide au calcul n'existe pour ce type de traitement, tout était réalisé manuellement, bref un traitement relativement lent et fastidieux mais impliquant des personnes à tous les niveaux.
- ***2ème : Automatique.*** Les procédés de business impliquent un traitement des données appliqué à la gestion de l'information. Tout ceci est souvent basé sur des processus manuels existants, simples et automatisés.
- ***3ème : Centré sur les bases de données.*** Le développement des technologies des bases de données permet la spécification des informations relatives au business.
- ***4ème : Outils courants.*** Les processus sont envisagés à deux niveaux de granularité différents : comme unités de travail qui sont regroupées grâce à un outil de workflow et comme implémentation de ces mêmes unités de travail via des applications spécifiques. Les workflows restent toujours compliqués à la fois pour le design et le redesign et fournissent toujours très peu de support pour le traitement des exceptions.
- ***5ème : Les Systèmes Multi-Agents.*** Cette génération émerge à travers l'utilisation de systèmes multi-agents. C'est de cette génération dont nous allons parler et dont nous allons étudier les spécificités mais aussi surtout les avantages qu'elle peut offrir en comparaison des générations précédentes.

FIG. 6.3 – Les générations des technologies du Workflow Management.

Nous voyons donc bien dans quelle direction se dirige les technologies du workflow management et nous allons tâcher dans le prochain chapitre de faire la lumière sur cette combinaison workflow management et systèmes multi-agents qui s'annonce très prometteuse.

Chapitre 7

Systèmes Multi-Agents et Workflow Management

Sommaire

7.1 Introduction

Grâce au chapitre précédent, nous avons rempli deux objectifs : d'une part, nous avons introduit le concept de workflow management et d'autre part nous avons pu, à la fin de ce chapitre 6, mettre en évidence les espoirs que nous plaçons

dans les systèmes multi-agents. En effet, nous voyons en ces systèmes un fantastique support pour les technologies du workflow management.

Pourquoi ? Les workflows sont présents partout au sein des entreprises au niveau de leur informatique de business. Ils n'entrent plus uniquement en jeu à l'intérieur de l'entreprise mais de plus en plus, entre les entreprises [SH99]. Bien que le rôle important qu'ils jouent dans la compréhension et l'automatisation des métiers des entreprises ne soit plus contesté, les pratiques actuelles de workflow management laissent encore à désirer. En effet, la **rigidité des technologies** du WM actuelles n'est pas en accord avec les **environnements dynamiques, complexes, hétérogènes** de nos jours. Heureusement, nous pensons que les systèmes multi-agents peuvent considérablement aider les technologies du WM à faire face à leurs environnements.

Au cours du prochain chapitre, nous mettrons en avant :
- les nombreuses correspondances qu'il existe entre ces deux technologies ;
- la complémentarité évidente qu'elles affichent ;
- les possibilités d'intégration de ces deux technologies ; et enfin,
- la nécessité de l'existence de modèles de références, de méthodologies et d'outils de développement.

7.2 Workflows et Agents, des caractéristiques communes

Pour le moment, notre idée est simplement de mettre en avant les différentes caractéristiques respectives des agents et des workflows et de la sorte, commencer à démontrer que l'**idée de les associer** s'inscrit dans une certaine logique.

7.2.1 Les Workflows

Tout comme les agents, les workflows ont certaines caractéristiques clés. Les workflows sont [SH99] :
- ***Composites***. Ils sont composés d'éléments différents, les tâches, dans notre cas.
- ***Complexes***. Leurs exécutions peuvent — ainsi que les activités qui les composent — être longues et sujettes aux défaillances, aux problèmes. Les workflows mettent souvent à jours des données, qui peuvent être accédées par des ressources différentes et les activités des workflows peuvent exiger de subtiles contraintes au niveau de leur cohérence.
- ***Coopératifs***. Ils n'impliquent pas uniquement une interaction humaine mais également des interactions entre les activités qui les composent.
- ***Hétérogènes***. Les environnements des workflows sont souvent hétérogènes et composés à la fois de nouveaux et d'anciens systèmes (appelés généralement *«legacy systems»*) impossibles ou presque à remplacer. Cette caractéristique joue en défaveur des systèmes de WM actuels qui ont souvent du mal à intégrer ces *legacy systems*.

- ***Autonomes***. Idéalement, les composants des workflows le sont mais là encore, les systèmes actuels ne peuvent la plupart du temps pas assurer cette autonomie de la façon la plus performante qui soit.

7.2.2 Les Agents

Comme nous l'avons étudié en détail au cours du chapitre 1, les agents possèdent des spécificités, des particularités qui les distinguent des paradigmes de programmation précédents. Les *agents* sont :

- ***Autonomes***. En plus d'être autonomes, et ce qui est finalement très intéressant dans notre cas, ils sont capables de nuancer cette autonomie pour se coordonner avec d'autres agents et participer ainsi activement à des systèmes multi-agents.
- ***Intelligents***. Ils peuvent apprendre, tirer profit de cet apprentissage et faire preuve de comportement intelligent lorsqu'ils agissent par exemple au nom d'un utilisateur.
- ***Sociables***. Ils sont acteurs du système et nourrissent des contacts au moyen soit de communications avec d'autres agents soit d'échanges interactifs avec des utilisateurs.
- ***Hétérogènes***. Ils doivent l'être afin de mieux s'adapter aux environnements dans lesquels ils sont actifs. C'est pourquoi, comme nous l'avons vu, il existe différents types d'agents inspirés de divers modèles : le modèle BDI inspiré de la psychologie et le modèle « *Commitments-Teams* » inspiré des organisations et sociétés ne sont que deux exemples parmi d'autres.
- ***Concurrents ou Coopératifs***. Bien qu'ils puissent se montrer concurrents si la situation l'exige (des agents peuvent par exemple représenter deux *brokers* sur un marché boursier, chacun essayant de tirer un avantage optimal lors de la réalisation d'une transaction), c'est surtout leur comportement coopératif qui est très intéressant. Travaillant ensemble, chacun des agents peut contribuer à la réalisation d'un objectif final.
- ***Complexes***. Un système multi-agents entier peut être vu comme un agent monolithique. Vu de cette façon, les agents qui le composent sont structurés entre eux, suivant une certaine décomposition hiérarchique. Les agents sont donc tout à fait adaptés pour implémenter des systèmes complexes et peuvent se montrer eux-mêmes très complexes.

Pour être complet, il nous faut ajouter encore quelques éléments. Tout d'abord, les agents peuvent fournir des accès aux ressources sous des contraintes précises, garantir des exigences d'intégrités élevées et répondre à des modèles d'organisation bien précis [SH99].

Ensuite, ils peuvent créer, construire ou constituer des « *mini-sociétés* » correspondant à des processus du business différents. Dans ce cadre, les agents prennent en charge la résolution des conflits qui pourraient survenir parmi les processus. En conséquence, l'utilisation d'agents est d'autant plus pertinente qu'il s'agit d'obtenir de la flexibilité, d'améliorer l'efficacité des processus et d'aider à la gestion de

la complexité.

Finalement, systèmes multi-agents et workflows insistent, tous deux, sur trois thèmes majeurs :

1. ***la largeur d'esprit*** caractérisée par des environnements où les adhésions et les comportements varient dynamiquement ;
2. ***le contrôle local*** afin de préserver les intérêts des designers et propriétaires des workflows ;
3. ***la cohérence*** de comportement qui aboutit en une cohérence globale.

Ceci mène à une correspondance naturelle entre les deux domaines scientifiques et l'existence de plusieurs synergies. Nous avons énoncé les avantages que possèdent les agents et dont les workflows pourraient bénéficier. L'inverse est également applicable. Certaines techniques de coordination des workflows pourraient naturellement être appliquées dans la coordination des agents. Sans plus tarder, nous allons voir dans quelle mesure les systèmes multi-agents et les workflows peuvent être **complémentaires**.

7.3 Une évidente complémentarité

7.3.1 Vers une combinaison de technologies

Après avoir introduit les *Technologies de l'Information* en leur sein pour améliorer l'efficacité de leur travail [YMS01], beaucoup d'entreprises se sont pourtant rendu compte que les processus business à l'intérieur de leur organisation ou entre elles et leurs partenaires n'ont pas été clairement définis, dessinés. En effet, des techniques et méthodes de suivi et contrôle des processus manquaient à l'appel ce qui se traduisait en une mauvaise compréhension des responsabilités, la création de blocs dans la coordination et des réactions trop lentes aux changements du marché.

Le Workflow Management est une de ces technologies en développement qui promet de résoudre en partie ces imperfections en fournissant une méthode de modélisation et de contrôle des processus business dans et entre les organisations. Mais, pendant que ces technologies se concentrent sur la *gestion de la logique des processus*, elles ne s'expriment pleinement qu'en **intégrant d'autres technologies** pour *contrôler complètement le processus de business* dans, par exemple, ses affectations de tâches et son allocation de ressources. Par rapport aux technologies de WM, les technologies liées aux agents fournissent flexibilité, distributivité et solutions intelligentes pour la gestion des processus business. La **combinaison de ces deux technologies** nous semble donc idéale pour fournir des solutions aux problèmes qui ne peuvent être résolus par aucune des deux séparemment.

Comme nous l'avons vu auparavant, une définition largement acceptée du workflow provient de la *Workflow Management Coalition* [Coa94] :

« Workflow is the computerized facilitation or automation of a business process, in whole or part. »

Les bénéfices de l'utilisation des technologies du workflow à la gestion des processus business sont nombreux :

- les processus business sont clairement définis. Les responsabilités et les relations de coordination le sont donc aussi ;
- il est facile d'optimiser les processus business puisqu'ils sont bien définis ;
- les processus business sont modularisés et par conséquent facilement réorganisables par le WFMS *(ou Workflow Management System)* ; et enfin,
- le WFMS peut suivre les opérations journalières, intégrer les applications sur des plates-formes différentes et séparer la logique du business des tâches elles-mêmes du processus. L'utilisateur doit uniquement se concentrer sur les tâches, pas sur le routage des processus de business.

Voyons maintenant sans plus tarder le revers de la médaille puisqu'il est évident que les systèmes de WM actuels, bien que de plus en plus utilisés, sont entachés d'une série d'imperfections.

7.3.2 Les lacunes du WM dans le Business Process Management

D'un point de vue technique, les systèmes de workflow management (WFMS) apportent des principes, méthodologies et technologies de différents domaines des sciences informatiques et de gestion, à savoir les systèmes de gestion de bases de données, client-serveur, distribués et hétérogènes, les GUI (Graphical User Interface), les systèmes de messagerie, la gestion des documents, la simulation et les pratiques de business et re-engineering.

Cependant, les générations actuelles de WFMS montrent des imperfections :

- ***Basé sur une unité centrale*** : les WFMS ont un serveur de workflow central qui définit et contrôle tous les processus de business. Ce type d'architecture ne convient évidemment pas aux entreprises distribuées et aux grandes compagnies.
- ***Manque d'automatisation*** : les WFMS déterminent la logique de processus mais la plupart des activités sont toujours réalisées par l'homme. Les WFMS ne sont pas capables de démarrer un workflow sans l'intervention humaine.
- ***Manque de réactivité [Tra96, OW98]*** : les WM ont besoin de connaître préalablement une représentation prédéfinie des processus de business qu'ils devront gérer et toutes déviations potentielles de ces scénarii.
- ***Manque de gestion des ressources [Tra96, OW98]*** : les WFMS ne contrôlent pas les ressources d'un processus de business, et comptent donc sur un processus dimensionné d'avance.
- ***Manque de sémantique [Tra96, OW98]*** : les WFMS n'ont aucune idée du contenu du processus et ne prennent pas de décision sur base de la nature de cette information.
- ***Manque d'interface générique*** : les WFMS ont besoin d'échanger des données entre les activités et de communiquer avec d'autres applications au

moyen d'interfaces. Actuellement, ces opérations dépendent d'appels API.

- ***Manque d'interopérabilité [OW98]*** : les WFMS sont développés par des vendeurs indépendants. Bien que le WfMC[1] ait publié ses travaux (des modèles de développement à suivre) pour permettre une interopérabilité entre les différents systèmes, nous sommes encore loin de solutions adaptées.

La solution à cette série d'imperfections est pour le WFMS de **bénéficier des mécanismes d'autres technologies**. Par exemple, intégrer le *project management* pour la gestion des ressources et la planification ou d'intégrer les *groupware* (défini au point 6.4, page 73) pour l'échange de messages et les environnements de travail personnel. La technologie des agents software est une de celles dont les technologies des workflows peuvent bénéficier le plus.

7.3.3 Les bénéfices de la technologie Agent dans la gestion des processus business

La techonologie Agent fournit une extension et une alternative aux problèmes de *Business Process Management*. Parmi les nombreuses propriétés que les agents possèdent et que nous avons par ailleurs déjà étudiées au point 7.2.2 de ce même chapitre mais également au cours du chapitre 1, les aspects autonomes, collaboratifs et intelligents sont les plus intéressants dans les systèmes distribués de *Business Process Management*.

Les bénéfices de l'application de la technologie agent au *Business Process Management* incluent :

- ***Architecture de système distribué*** : la technologie agent fournit une structure de système distribuée qui facilite l'intégration de plusieurs systèmes de *Business Process Management* distribués.
- ***Automatisation*** : l'autonomie des agents software leur permet de réaliser des activités comme substituts de l'homme.
- ***Interaction*** : les agents software permettent aux organisations d'interagir.
- ***Gestion des ressources*** : les agents peuvent représenter des ressources. L'assignation des tâches et l'allocation des ressources se fait par négociation entre les agents.
- ***Réactivité*** : les agents réagissent aux circonstances changeantes et ont la capacité de générer des scénarii d'exécution alternatifs. Cette capacité fait usage de l'intelligence des agents et de leur apprentissage.
- ***Interopérabilité entre les systèmes hétérogènes*** : les agents peuvent être hétérogènes. Les interactions se reposent sur des messages sémantiques pour l'échange de plans et de services. Cela rend l'interopérabilité plus facile que les appels API.
- ***Décision intelligente*** : certaines caractéristiques de haut niveau des agents, telles que l'apprentissage, sont aussi très pratiques au workflow management

[1] Pour rappel, il s'agit de la *Workflow Management Coalition*, `http://www.wfmc.org`, l'organisation internationale composée des vendeurs, utilisateurs, analystes et groupes de recherche ou universitaires pour la définition des standards du workflow.

bien qu'elles ne soient pas encore tout à fait matures.

Cependant, et nous voulons insister sur ce point, l'implémentation de systèmes de gestion de processus de business en utilisant uniquement la technologie des agents souffre des problèmes suivants :

- un *mécanisme de coordination* entre les tâches appartenant aux différents flux de travail manque à l'appel, ce qui peut rendre les systèmes instables et non fiables ;
- l'*optimisation des processus de business* est difficile suite au manque de définitions explicites et de représentations des processus de business ; et enfin,
- il n'est pas facile de *tracer les opérations journalières* contrairement aux systèmes de workflow management.

Le lecteur l'aura bien compris à travers ce point et le point précédent, notre objectif a été de démontrer la complémentarité affichée par les deux technologies et les avantages que le workflow management pourrait tirer de cette combinaison. Pour nous, il est évident que chacune des deux technologies peut répondre aux lacunes de l'autre et ainsi, participer à l'élaboration d'un système de workflow management répondant totalement à ses exigences.

Bien que nous soyons désormais convaincus, et nous espérons que le lecteur l'est également, notre étude ne s'arrête évidemment pas ici. La prochaine étape sera en effet pour nous d'étudier de quelles manières ces deux technologies peuvent s'*intégrer* dans un même système. Cette question fera l'objet du prochain paragraphe.

7.4 Intégration des deux technologies

Récemment, des vendeurs de systèmes de workflow management ont affirmé qu'ils avaient implémenté des agents software dans leurs produits commerciaux de gestion de workflow [YMS01]. Il s'avère cependant que ces agents sont très basiques et se réduisent souvent à l'implémentation d'activités autonomes. Comme beaucoup de chercheurs, nous pensons que la technologie agent peut fournir des architectures de système facilitant l'intégration de multiples systèmes de workflow. Comme nous allons le montrer, l'application des agents aux systèmes de workflow management peut se faire de deux manières différentes [YMS01] :

1. Le système de workflow management est *amélioré* qualitativement par la présence des agents ;
2. Le système de workflow management est *basé* sur les agents.

7.4.1 Un Workflow Management amélioré par la présence des Agents

Cette première possibilité d'intégration des deux technologies est la forme la plus *basique* pour appliquer les agents au workflow management. La figure 7.1

nous permet de mieux comprendre l'architecture d'une telle intégration. Dans ce cas de figure, un **moteur de workflow central** contrôle toutes les activités. Les agents sont invoqués pendant l'exécution d'un travail pour implémenter certaines tâches. Le système de workflow contrôle la **génération et l'élimination des agents**. Cette architecture correspond bien à la façon dont les systèmes de WM commerciaux actuels utilisent le concept d'agents. Les agents peuvent accomplir les tâches suivantes :

- ***Interface avec l'utilisateur*** : il s'agit là d'une partie de l'environnement de travail que le système de workflow fournit à ses utilisateurs. L'agent agit en tant qu'assistant personnel. Trier les e-mails, répondre à un e-mail, rappeler les événements ne sont que quelques exemples. Ce type d'agent est représenté par l'agent ***A1*** dans la figure 7.1.
- ***Activité autonome*** : l'agent implémente les tâches de façon autonome sans intervention humaine quelle qu'elle soit. ***A2*** dans la figure 7.1.
- ***Interface avec les autres applications*** : l'agent peut servir d'interface vers d'autres applications. C'est le cas des agents ***A6-A7*** dans la figure 7.1. Ceci est très intéressant, sachant que les systèmes de workflow management doivent souvent s'interfacer et collaborer avec des systèmes plus anciens qui continuent à fonctionner (car indispensables). Ces anciens systèmes sont appelés *« legacy systems »* [Eng]. Le système entier est donc très hétérogène. Au lieu de définir des API pour l'interopérabilité entre les applications, des formats de messages sémantiques (autrement dit des messages dont le *sens* est pleinement compris par les différents agents) peuvent être définis afin de permettre l'échange d'informations de haut-niveau. Cela ouvre également les portes de la construction d'interfaces génériques pour les applications.
  Entre les agents ***A6-A7*** et les applications *legacy* invoquées figurent des *wrapper*[2] d'objets [Eng]. Les *wrapper* permettent l'échange d'informations entre le système de WM et les anciennes applications en emballant ou déballant les données dans des structures adaptées lisibles soient par le système de WM soit par les systèmes *legacy*.

Dans ce scénario, les agents peuvent être considérés comme des *services* fournis par le système de workflow management. En utilisant des agents, l'objectif est d'**augmenter l'automatisation** du système de workflow. Nous l'avons vu, les agents peuvent améliorer l'autonomie, la communication et la réactivité du système. En se substituant à l'homme, les agents réalisent nombre de tâches sans aucune intervention humaine.

D'un point de vue plus systémique, les agents n'interagissent malheureusement pas les uns avec les autres. En réalité, le *moteur de workflow* contrôle leurs actions. Toute information échangée se fait à travers ce moteur qui est le seul responsable de la *création* et de la *suppression* des agents. De plus, dans ce cas de figure, les agents ne sont pas nécessairement intelligents. En d'autres mots, ils n'affichent peut-être pas clairement les importantes notions des agents comme les aptitudes mentales (modèle BDI). Dans la plupart des produits de WM, l'agent est en

[2] *Wrapper* signifie littéralement papier d'emballage, couverture.

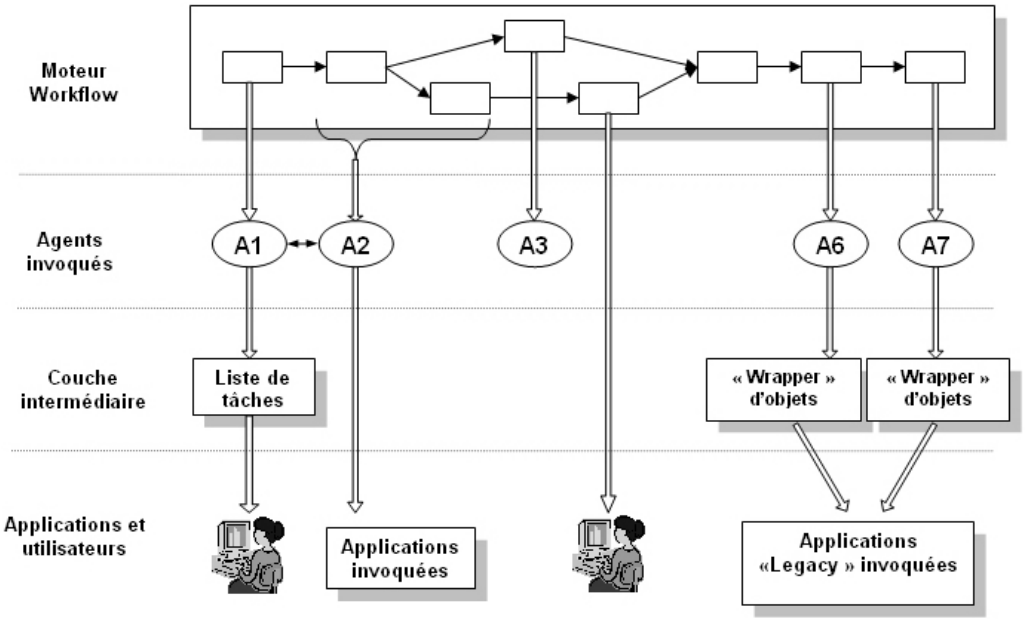


FIG. 7.1 – Système de Workflow Management amélioré par les Agents [YMS01].

quelque sorte **un morceau de software ordinaire**, comme par exemple dans MQSeries Workflow [IBM00] et InConcert [InC00].

7.4.2 Un Workflow Management basé sur les Agents

Dans ce deuxième cas de figure, le système de workflow est *basé* sur les agents et comme nous pouvons nous en rendre compte à travers la figure 7.2, il consiste en **un système distribué de multiples agents**. Les différents agents sont indépendants les uns des autres et chacun est responsable de la bonne exécution du processus. Dans ce scénario, le processus de business entier est découpé en petits morceaux, en petits sous-réseaux. Chacun de ces petits sous-réseaux se retrouve au sein d'un des différents agents du système (Agent 1 à 4 dans la figure 7.2). Tout le corps, la logique du processus est donc embarquée dans les agents, au lieu d'être explicitement représentée ailleurs. **C'est ce scénario d'intégration des deux technologies qui nous a servi de guide dans la réalisation de notre application.** Comme nous venons de l'avancer, le corps de notre business se trouvera donc à *l'intérieur de nos différents agents.*

Ce scénario est très intéressant quand il s'applique au monde réel et aux contraintes actuelles des entreprises : les processus de business actuels couvrent souvent plusieurs unités dans une entreprise et même parfois plusieurs compagnies. Ce « fractionnement » du business rend le contrôle par un moteur central de workflow impossible ou peu performant. Le moteur est en effet incapable d'obtenir toutes les informations requises du business entier. Une solution plus performante et répondant mieux aux contraintes des environnements actuels est de disposer d'**un moteur de workflow résidant dans chaque organisation ou unité**. Le

processus de business est ainsi réalisé dans son intégralité grâce aux interactions entre les différents moteurs de workflow.

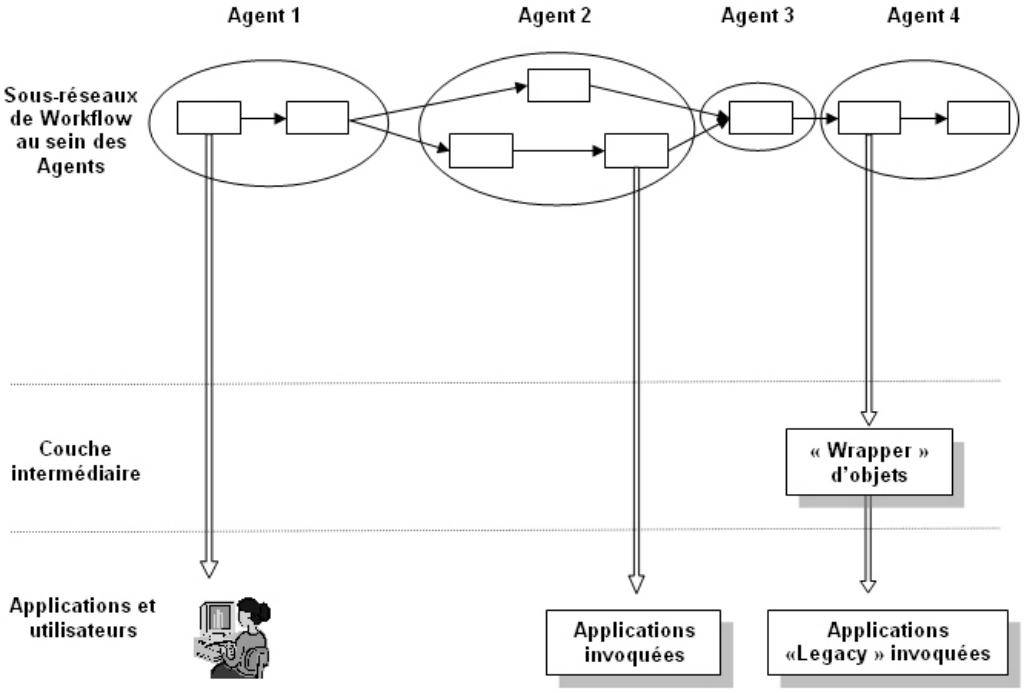


FIG. 7.2 – Système de Workflow Management basé sur les Agents [YMS01].

Dans ce cas de figure, les agents ont l'entière responsabilité du système de WM. Concrètement, les agents :

- disposent de tous les moyens pour ***analyser***, ***automatiser***, ***intégrer*** et ***inspecter*** les workflows ;
- ***communiquent*** et ***interagissent*** les uns avec les autres ; et enfin,
- font bénéficier le système de leurs ***compétences***, leurs ***aptitudes*** de haut-niveau telles que l'apprentissage, la négociation qui se veulent à l'heure actuelle très prometteurs même s'ils font encore l'objet de recherche et ne sont pas encore matures.

Prenant en charge le bon déroulement du processus de workflow dans son intégralité, les agents deviennent naturellement plus complexes que dans le premier cas de figure. En plus de l'autonomie, de la communication et de la réactivité, les agents sont désormais :

- ***cohérents*** : ils sont homogènes et font preuve d'une certaine cohésion, adhèrent les uns aux autres et sont unis dans la réalisation de leurs tâches.
- ***orientés objectifs*** : ils agissent en réponse à des stimuli et tentent par tous les moyens mis à leur disposition d'atteindre leurs objectifs. Cette caractéristique fera l'objet de notre prochain chapitre. En effet, comme nous l'avons vu lors du chapitre 1, il existe différents types d'agents dont les deux prin-

cipaux sont les agents orientés événements[3] et les agents orientés objectifs[4]. Durant la réalisation de notre système, nous avons utilisé dans un premier temps les événements pour diriger les agents puis par la suite, nous avons tenté de n'utiliser que des agents orientés objectifs, principalement pour les tâches et les processus de workflow management, et pour des raisons de performance et de capacité à accomplir ceux-ci. Selon nous, l'utilisation d'agents orientés objectifs peut être très intéressante pour la réalisation de processus de workflow management. Ainsi, toujours selon nous, le fait qu'ils tentent par tous les moyens d'accomplir les tâches qui leur sont attribuées est un comportement très proche de celui que tout membre d'une organisation, tout participant à la réalisation d'une tâche ou d'une activité reproduit ou devrait reproduire.

Dans ce deuxième scénario, l'utilisation des agents est bénéfique au workflow management en ce sens que :

- ils permettent de ***définir une architecture distribuée du système*** ;
- ils sont munis de ***méthodes et de langages de communication*** agents tels que le KQML, langages qui permettent une interopération des systèmes de workflow, spécialement pour les systèmes hétérogènes ;
- ils sont autonomes et permettent ***le fonctionnement automatique du système***, puisqu'ils ont la capacité, une fois les objectifs attribués, d'exécuter les tâches sans aucune participation « humaine ».
- ils s'***ajustent à l'environnement***, sont capables de modifier leurs plans et de créer de nouvelles voies de fonctionnement ;
- ils possèdent des caractéristiques de haut-niveau comme l'apprentissage, la négociation et surtout la ***planification*** qui seront à l'avenir très prometteuses.

De plus, au delà de l'aide à l'automatisation des processus business, les agents permettent également de réduire indirectement les coûts : le système ne requiert plus une grande participation humaine, elle se voit réduite. Cette diminution de l'implication des utilisateurs a un impact considérable sur les coûts totaux des processus de business. De plus, le temps de réponse à des changements environnementaux est revu à la baisse et l'interopérabilité entre des systèmes hétérogènes, nous insistons à nouveau sur ce point, est considérablement améliorée.

A l'heure actuelle, très peu de produits commerciaux de workflow management implémentent ce type d'architecture et de fonctionnement mais les recherches sont nombreuses et attirent bien des attentions. En effet, ces systèmes de workflow management basés sur les agents attirent l'attention sur l'étude des interactions entre des systèmes à multiples workflows, des ***systèmes multi-workflows***, qui seront certainement les exigences de la prochaine génération de systèmes de workflow.

Ces recherches portent notamment sur les différents points suivants : l'architecture du système, les procédures et protocoles de négociation entre les agents, la

[3] *event-driven* ou *event-directed* en anglais
[4] *goal-driven* ou *goal-directed* en anglais

gestion des ressources, la planification des tâches et des activités qui est une des fonctionnalités majeures de notre application, la modélisation et les architectures agents. Nous reviendrons sur les deux premiers points lors de l'analyse de notre application. Dans la suite de ce chapitre, nous allons tenter de dessiner, façonner, construire une méthodologie à suivre pour le développement d'un système de workflow management basé sur des agents et plus particulièrement même, sur des agents orientés objectifs qui, pour rappel, feront l'objet du prochain chapitre.

7.5 Une méthodologie de développement pour les systèmes de WM basés sur des agents

A la lecture de ce qui précède, le lecteur sera certainement convaincu que les technologies multi-agents sont bien celles qui permettront de mettre sur pied des systèmes de workflow management de la meilleure façon qui soit au coeur des entreprises ou de toute autre organisation. Après l'avoir étudié au point précédent 7.4.2, nous partirons de l'idée -nous en sommes convaincus- que ce deuxième scénario représente le futur de l'intégration entre les technologies du WM et des SMA.

Cependant, comme n'importe quel autre système, un système de WM basé sur des agents n'échappe pas aux besoins de l'existence d'une méthodologie de développement claire et précise. L'existence d'une méthodologie et de modèles est primordiale à toute organisation pour deux raisons très simples : elle est synonyme de réduction des coûts (en argent, en temps, en ressources, en effort, etc) et elle augmente la vitesse de développement qui n'est plus à négliger à l'heure actuelle. Cette méthodologie devra combler à la fois les besoins de modélisation des processus de workflow management mais devra aussi permettre la modélisation du système multi-agents. Chacun des deux aspects devra être pris en compte et même plus, ils devront être modélisés et étudiés ensemble. Malheureusement, à l'heure actuelle, il est très difficile de trouver une méthodologie de développement qui satisfasse ces exigences. Dans un premier temps, nous allons voir rapidement les différentes représentations qu'il est possible de faire à la fois d'un système de workflow management et d'un système multi-agents pour savoir ce qu'il y a lieu de modéliser. Dans un second temps, nous tenterons de construire une méthodologie de développement de système de WM basé sur des agents en partant de méthodologies soit spécifiques au workflow management, soit spécifiques aux systèmes multi-agents.

7.5.1 La spécification d'un Workflow

Dans leur article *« Multiagent Systems for Workflow »* [SH99], Munindar P. Singh et Michael N. Huhns parlent « d'abstractions de modélisation » utilisées pour capturer les différents composants ou aspects d'un système d'information et plus particulièrement d'un CIS[5] ou SIC en français pour ***Systèmes d'Infor-***

[5] *Cooperative Information Systems*

mation Coopératifs. Les systèmes SIC sont des systèmes multi-agents avec des abstractions organisationnelles et de bases de données particulièrement bien adaptées aux environnements ouverts et complexes actuels. Pour faire le lien avec les workflows, nous définissons simplement les workflows comme les aspects dynamiques des SIC. Plus précisément, un workflow est une spécification de certaines classes de comportement d'un SIC.

Distinguons dès à présent les deux types de modèles existants :

1. ***Les modèles statiques*** : tels les modèles conceptuels et organisationnels, ils sont également indispensables aux workflows puisque finalement, tout système de workflow doit s'occuper de ces différents aspects.
2. ***Les modèles dynamiques*** : les modèles transactionnels et les diagrammes de flux de contrôle ne sont que deux exemples de modèles dynamiques, modèles dynamiques qui ont un impact direct sur le workflow management.

La figure 7.3 qui suit nous montre les différents modèles utilisés lorsqu'il s'agit de modéliser un workflow dans une organisation ou une entreprise.

Parmi les différents modèles, nous retrouvons les deux suivants :

- le ***schéma Entité-Assocation*** qui est un modèle conceptuel de l'information stockée dans l'organisation ;
- la ***décomposition des activités*** qui décrit la relation d'inclusion entre elles alors que les diagrammes de flux de contrôle, de données et de matériels donnent des informations supplémentaires à leur sujet.

Le schéma E-A correspond à de l'information statique comme des ontologies alors que les représentations des activités correspondent plus ou moins aux workflows. Les deux catégories de représentation sont importantes puisque les actions dans les workflows dépendent des concepts qu'ils manipulent et les concepts sont définis sur base des modèles d'utilisation.

La structure organisationnelle d'un SIC, à savoir l'ensemble des rôles et des responsabilités qui font que le système fonctionne, est très importante. Comme nous le montre la figure 7.4 (page 95), il existe une relation entre les workflows s'exécutant dans un SIC et les rôles organisationnels disponibles dans celui-ci. Dans cet exemple, la partie de gauche représente un simple workflow. La tâche *« Write White Paper »* y est décomposée en plusieurs sous-tâches. Certaines tâches du workflow affectent des bases de données et/ou d'autres processus sous-jacents. Encore plus important pour nous, nous allons voir pourquoi les tâches sont liées à la structure organisationnelle de la compagnie car les étapes clés du workflow sont accomplies, réalisées par des personnes munies d'autorités spécifiques.

C'est ici que peuvent intervenir les agents en faisant valoir leurs innombrables qualités. Traditionellement, les rôles sont associés aux tâches de façon rigide. Cependant, dans les environnements complexes et dynamiques que nous connaissons aujourd'hui, des liens plus flexibles sont requis. En effet, que se passe-t-il dans le système si un acteur est absent ? Comment le processus de workflow peut-il continuer, de quelle façon, par quel(s) moyen(s) ? De plus, si une même personne

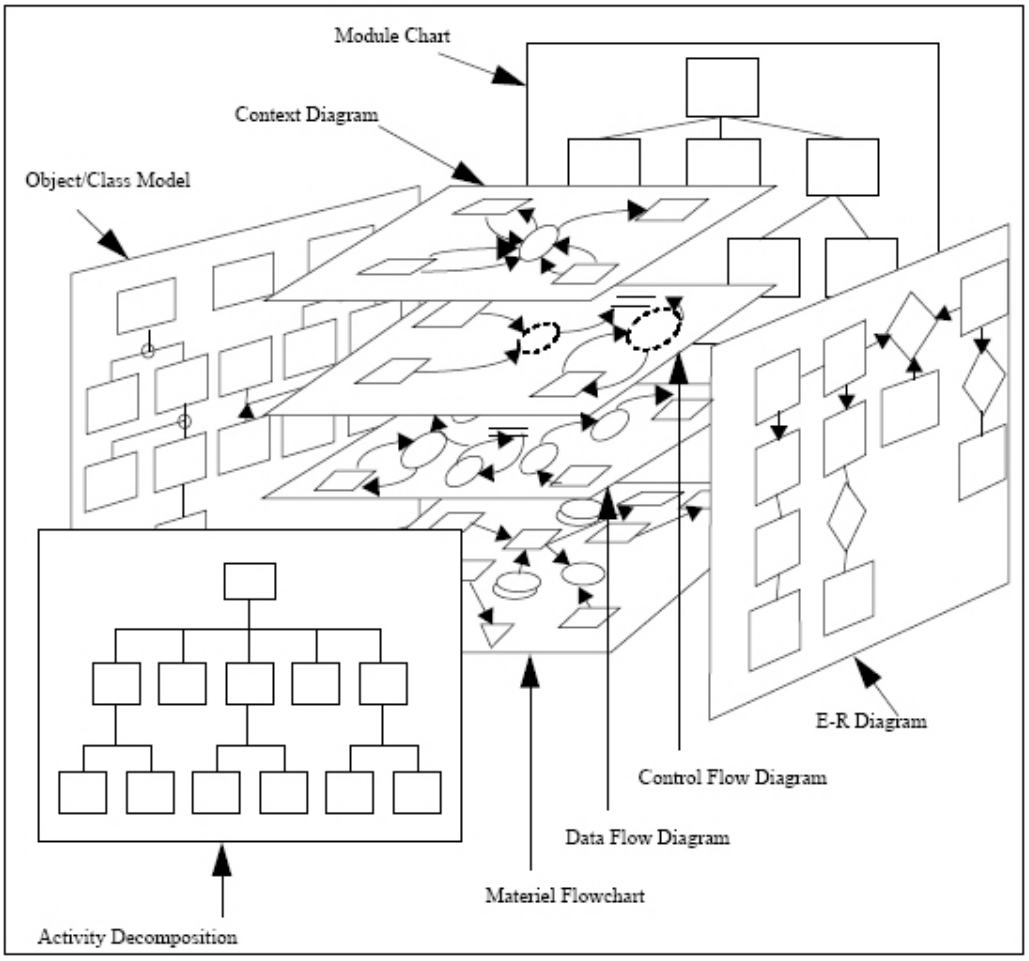


FIG. 7.3 – Modèles utilisés pour décrire un SIC [SH99].

remplit plusieurs rôles, comment le workflow en tient-il compte pour optimiser le temps d'occupation de cette personne ? Enfin, comment procéder pour faire correspondre les obligations d'une organisation aux obligations et actions de ses membres et comment les décisions d'un membre peuvent-elles être rejetées, annulées et ses actions défaites ? Réciproquement, comment un participant peut-il obtenir les autorisations et les pouvoirs nécessaires, qui représentent des exceptions aux politiques par défaut, afin de répondre proprement à une situation inattendue. La réponse à tout ceci est très simple : l'utilisation des agents permettra de répondre à toutes ces questions grâce aux nombreuses qualités qu'ils possèdent. Nous étudierons cela au point suivant (point 7.5.2) mais il s'agit bien là d'une raison supplémentaire pour disposer d'une méthodologie qui nous permettra de modéliser à la fois le workflow management et le système multi-agents sur lequel il sera basé et qui permettra d'optimaliser son exécution.

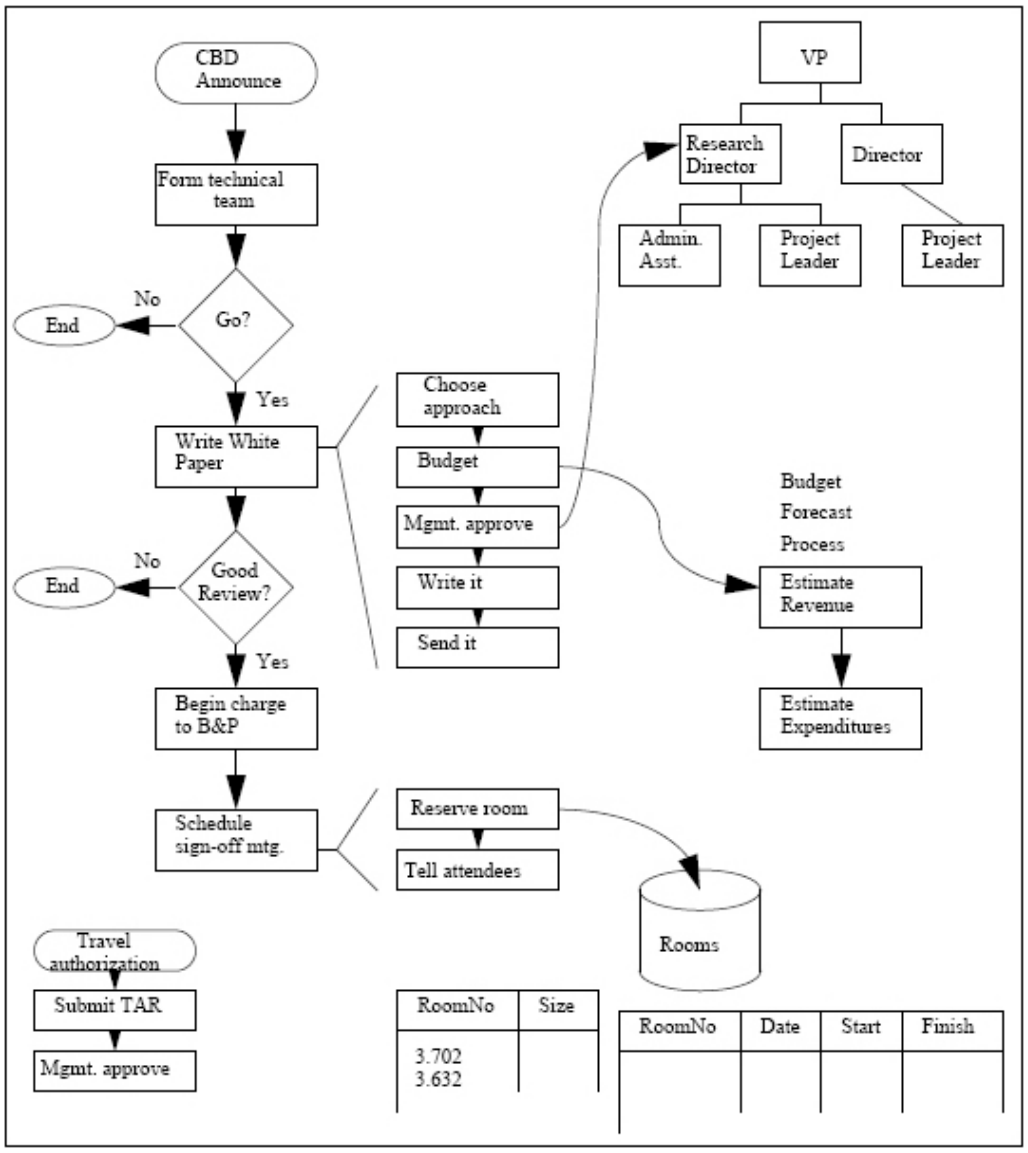


FIG. 7.4 – Exemples de relations entre les différents modèles [SH99].

7.5.2 Les apports supplémentaires des agents à prendre en compte lors de la spécification

A côté de toutes les qualités des agents que nous avons déjà maintes fois mentionnées au cours du chapitre 1 et du présent chapitre, d'autres caractéristiques sont à mettre en avant puisqu'elles sont en relation de près ou de loin avec le domaine du workflow management. Elles sont notamment mises en évidence lors de la spécification du workflow.

Ainsi, nous avons vu au point précédent les nombreuses difficultés qui peuvent survenir lors de la construction d'un workflow suite à la complexité et la dynamique des environnements actuels. Jusqu'à maintenant, des questions restent

ouvertes en raison des limites que possèdent les récents systèmes de workflow management. Au point précédent, nous avons listé une partie de ces nombreuses questions qui restent toujours en suspens. A l'instar des spécialistes du domaine, nous pensons que les agents vont résoudre ces nombreux problèmes et permettre aux systèmes de workflow management :

- d'être performants dans le ***traitement des tâches habituelles*** ;
- de ***répondre à une plus grande diversité de situations*** ; et surtout,
- d'être capables de ***faire face à des situations inattendues***.

Comme nous l'avons déjà dit, les agents sont ***autonomes*** et très efficaces dans des environnements ***hétérogènes***. Encore plus important à nos yeux au niveau du workflow management, ils permettent de ***modéliser des organisations*** et peuvent ***représenter des tâches « sans fin »***[6]. Ils peuvent aussi ***créer des « mini-sociétés »*** qui correspondent aux différents processus du business, mais conservent ***un certain degré de responsabilité*** afin de résoudre tout conflit qui pourrait survenir entre différents processus. Les agents offrent donc bien flexibilité, agilité et efficacité pour mieux gérer la complexité d'un système.

Au point 7.3.2, nous avons pu voir que les systèmes de workflow management actuels souffrent d'un manque d'autonomie, d'hétérogénéité pour ne citer que les principales lacunes. A côté de cela, il existe un réel besoin de « faire face à des situations inattendues », autrement dit le ***traitement des exceptions***. Les exceptions sont difficiles à prévoir, à prédire, lors de la conception et du design d'un système de workflow management. Traditionnellement, lorsqu'elles surviennent, seul l'homme peut les traiter. Cependant, avec l'utilisation des agents, il devient possible de leur attribuer cette tâche. Ils peuvent prendre cette responsabilité, sous la supervision de l'homme si cela est vraiment nécessaire. Pour rappel, lorsqu'un événement survient, cela induit souvent une réponse auprès d'un ou de plusieurs agents qui agissent en conséquence. Si l'exécution du premier plan échoue, l'agent est capable de considérer l'application d'un autre plan afin d'atteindre à tout prix ses désirs (objectifs). Le système de workflow management est alors enrichi de nouveaux processus, de nouveaux workflows et est désormais plus complet. Toujours dans le très intéressant article *Multiagent Systems for Workflow* [SH99], les deux auteurs estiment que la différence majeure entre le domaine expérimental et l'utilisation en production est essentiellement la complétude dans le traitement des exceptions. C'est pourquoi ils ajoutent que le design et la mise sur pied des workflows doivent être « interfoliés[7] ». Le design du système de WM doit laisser la porte ouverte à des extensions qui viendront enrichir le système plus tard, pour que celui-ci soit encore plus apte à répondre à la plus grande diversité possible de situations.

Imaginons une solution où chaque agent représente un acteur d'une organi-

[6] Un rôle, une responsabilité que l'agent possède toujours. Il ne réalise pas qu'un traitement en réponse à un événement mais réalise cette tâche continuellement.

[7] Brocher ou relier un livre en insérant des feuillets blancs entre les feuillets qui portent l'écriture ou l'impression.

sation. Grâce à ses aptitudes mentales, ses capacités d'apprentissage et le fait qu'il soit orienté objectifs, il peut se substituer à cet acteur pour jouer son rôle et prendre ses responsabilités. Il est orienté objectifs ce qui lui permet d'agir de la même façon que cet acteur, essayant d'accomplir une tâche en utilisant le moins de ressources possibles. Ses aptitudes mentales lui permettent de faire face à des situations inattendues et ses capacités d'apprentissage, d'emmagasiner un maximum d'informations pour que chaque nouvelle expérience, situation qui survienne régulièrement dans un environnement complexe soit pour lui une source enrichissante d'informations qui lui permettra, dans une prochaine situation de ce genre, de répondre encore plus vite et plus efficacement à cette situation.

Bien évidemment, il est nécessaire et primordial de pouvoir exprimer tout cela lors de l'analyse et la conception du système de workflow management basé sur l'utilisation d'agents. A l'heure actuelle, il existe des méthodologies de développement spécifiques aux systèmes de workflow management et spécifiques aux systèmes multi-agents mais il est presque impossible de trouver une méthodologie qui serait efficace lors de la construction d'un système qui serait une combinaison de ces deux technologies. Nous avançons deux solutions afin d'élaborer une méthodologie idéale :

1. adapter une méthodologie de développement de processus de workflow management aux systèmes multi-agents ; ou,
2. adapter une méthodologie de développement de système multi-agents au domaine du workflow management.

Etudions ces deux possibilités plus en détail.

7.5.3 Adaptation d'une méthodologie spécifique au Workflow Management

Le succès des projets de workflow dépend largement de la façon dont les processus de développement des applications du workflow sont planifiés, organisés et conduits. Dans leur article *« A Reference Model for Workflow Application Development Processes »* [WGHS99], les auteurs présentent un modèle de référence qui guide les managers et les développeurs à travers la structure complexe des processus de workflow et dont l'objectif est de développer des applications plus adéquates, plus efficaces et plus fiables. La figure 7.5 nous montre le modèle de référence pour les WADP[8].

1. ***Phase d'étude.*** Définir les objectifs du système, établir l'équipe du projet et rassembler toute l'information disponible. Un modèle de processus de business est choisi.
2. ***Phase de design.*** Le modèle choisi est développé, analysé et optimisé pour répondre aux objectifs initiaux.
3. ***Sélection du système.*** Choix d'un système de workflow management qui convient au modèle développé.

[8] *Workflow Application Development Processes* ou processus de développement d'application de workflow en français.

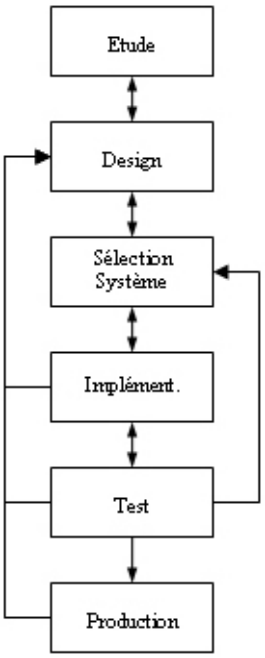


FIG. 7.5 – Modèle de référence pour *WADP* [WGHS99].

4. ***Implémentation.*** Implémentation du système.
5. ***Phase de test.*** Test du système avant son déploiement final et sa mise en production.
6. ***Phase opérationnelle.*** Mise en place définitive du système.

Le développement séquentiel est une situation idéale mais dans la plupart des cas, cela ne se passe pas comme cela. Afin de faire face à tout problème survenant lors du développement, certaines phases peuvent être effectuées plusieurs fois. C'est pourquoi dans la figure 7.5, il existe des flèches supplémentaires qui ne figurent pas dans un processus de développement séquentiel normal et qui traduisent bien le caractère itératif du développement envisagé.

Pourtant, bien qu'amélioré par cette approche itérative, cette méthodologie de développement reste très classique. A chaque étape, les modèles utilisés pour représenter le système sont totalement orientés workflow. Le modèle général peut être « customisé » dans plusieurs voies différentes (par exemple, certaines étapes seront rallongées ou raccourcies en fonction des compétences de l'équipe et des caractéristiques du projet et de l'environnement) mais elle ne permet pas pour autant de mieux prendre en compte la présence d'un système multi-agents.

En effet, quel que soit le modèle, chaque étape du cycle de vie requiert des outils efficaces basés sur des modèles et accompagnés d'une méthodologie. Mais comment faire pour intégrer les spécifications d'un système multi-agents durant ce cyle de vie ? Cela entraîne naturellement **une lourde redéfinition des modèles** pour prendre en compte les caractéristiques des agents. Dans leur article *« Towards*

Adaptive Workflow Enactment Using Multiagent Systems » [BV05], Paul A. Buhler et José M. Vidal proposent que les langages de description de workflow et leurs outils associés soient utilisés pour spécifier les systèmes multi-agents. Plus précisément, ils avancent l'idée que BPEL4WS[9][10] puisse être utilisé comme langage de spécification pour exprimer l'ordre social initial du système multi-agents. Leur proposition est très bien résumée par l'aphorisme suivant :

Moteurs de workflow adaptifs[11] = *Services Web* + *Agents*

Les services Web sont les ressources de calcul et les agents s'occupent de la coordination.

BPEL4WS se concentre sur l'expression du mécanisme du workflow alors qu'un autre langage, DAML-S est utilisé pour la sémantique. Leur vision consiste à utiliser les services Web comme comportements externes pour des agents proactifs. Les services Web ne connaissent qu'eux-mêmes, ils ne possèdent pas d'aptitudes mentales, ne sont pas capables d'actions autonomes, de communications intentionnelles ou encore de comportements coopératifs. A l'opposé, comme nous l'avons déjà vu, les agents possèdent toutes ces capacités. Les capacités sociales des agents sont essentielles et conviennent parfaitement aux développements de systèmes complexes. Ils peuvent former des structures sociales. Chaque agent s'engage individuellement dans la réalisation d'objectifs si bien que leurs interactions coordonnées permettent d'atteindre des objectifs plus globaux. Ils agissent alors comme une entité collective. Le problème principal est de donner aux agents des descriptions sémantiques de leurs comportements pour qu'ils puissent raisonner et être conscients de leurs capacités et de celles des autres agents présents dans le même système. Leur approche nouvelle consiste, comme nous l'avons déjà dit, à utiliser BPEL4WS pour imposer un ordre social initial à une collection d'agents. Puisque BPEL4WS décrit les relations entre les services Web dans le workflow, les agents qui représentent les services Web connaîtront leurs relations à priori. Nous laissons au lecteur le soin de consulter leur article [BV05] pour obtenir plus de détails à propos de cette solution.

«Models are not right or wrong, they are more or less useful...» [Fow97]

Cette solution paraît tout à fait réalisable. L'adaptation de la méthodologie pour mieux répondre aux caractéristiques des agents et mieux les prendre en compte nous semble possible. Cependant, nous ne sommes pas convaincus de la qualité dont pourrait faire preuve cette solution. Selon nous, elle limiterait fortement l'expression des comportements des agents, empêchant dès lors le système de WM de bénéficier de toutes leurs qualités, ce que nous voulons à tout prix éviter. En accord avec la citation de M. Fowler que nous avons mentionnée ci-dessus, nous affirmons que cette méthodologie ne serait pas mauvaise, mais elle ne serait pas

[9] *Business Process Execution Language for Web Services*

[10] C'est durant l'été 2002 qu'IBM, Microsoft et BEA sortirent ce nouveau PLD (*Process Description Language*) appelé BPEL4WS, standard *de facto* pour exprimer les workflows sous forme de services Web.

[11] Equivalents aux moteurs de workflow classiques à cela près qu'ils peuvent réagir aux conditions changeantes des environnements dynamiques et complexes actuels.

non plus très utile. Nous ne continuerons donc pas à explorer cette solution car nous sommes persuadés que l'adaptation d'une méthodologie de développement de système multi-agents sera une solution bien plus performante.

7.5.4 Adaptation d'une méthodologie spécifique aux Systèmes Multi-Agents

Les qualités que peuvent offrir les technologies agents aux systèmes de workflow management ne sont plus un secret, nous les avons mises suffisamment en évidence. Nous voulons maintenant démontrer que même si cette constatation semble indiscutable, il est toute autre chose de posséder les méthodologies et les outils adéquats pour parvenir à mettre de tels systèmes sur pied. Nous venons d'avancer que l'adaptation d'une méthodologie de développement spécifique au workflow management n'est pas la meilleure solution pour y arriver. Dans la suite de ce point, nous désirons démontrer pourquoi nous sommes convaincus que la deuxième solution, à savoir l'adaptation d'une méthodologie spécifique aux systèmes multi-agents, portera beaucoup plus ses fruits. Nous montrerons aussi quelques pistes qui pourraient être poursuivies pour y parvenir.

Nous avons vu au cours du point 7.4 qu'il existait deux solutions d'intégration de ces deux technologies : un système de WM amélioré par la présence d'agents et un système basé sur ceux-ci. Nous avons vu également que la deuxième solution, celle que nous avons adoptée pour réaliser notre projet, semble la plus efficace car elle permet d'exploiter au maximum les qualités, capacités et compétences des agents. Dans la même logique, nous affirmons qu'utiliser une méthodologie de développement spécifique aux agents ne limitera pas l'expression des actions et des comportements des agents, limite dont nous voulons absolument nous prémunir. Si nous prenons par exemple le cas de la méthodologie Prometheus que nous avons utilisée pour réaliser l'analyse et la spécification entière de notre projet, elle permet entre autre de prendre en charge certains modèles bien spécifiques aux agents comme le modèle BDI. Ainsi, dès la spécification, nous disposons de moyens pour mieux exprimer le comportement que le système et les agents devront adopter par la suite. Nul doute que l'expression des aptitudes mentales des agents intelligents et spécialement des agents orientés objectifs, le type d'agents que nous avons utilisé pour réaliser notre projet, serait une tâche lourde, fastidieuse, incomplète et sans doute sans succès avec les modèles et outils disponibles d'une méthodologie spécifique au workflow management. A la lecture du chapitre 4 consacré aux méthodologies de développement, le lecteur se sera certainement rendu compte des larges possiblités qu'offrent la méthodologie Prometheus quant aux spécifications des comportements, des actions, des aptitudes, des capacités, des interactions, des relations et autres éléments relatifs aux agents.

Reste qu'à ce niveau, le problème inverse au point précédent survient : il s'agit pour nous de spécifier les processus de workflow management lors de l'analyse du système. La méthodologie spécifique aux systèmes multi-agents doit être adaptée pour pouvoir satisfaire à cette condition. Pour y parvenir, nous proposons ci-dessous différentes pistes qui pourraient être suivies et plusieurs éléments dont il

faudra tenir compte.

Intégrer le modèle de référence des Workflows

Bien que publié en 1995 par le WfMC [Coa95], le modèle de référence du workflow définit toujours une architecture intéressante pour discuter des systèmes de workflow management au jour d'aujourd'hui. La figure 7.6 nous montre les composants fontionnels d'un système de workflow et identifie les principales interfaces entre eux.

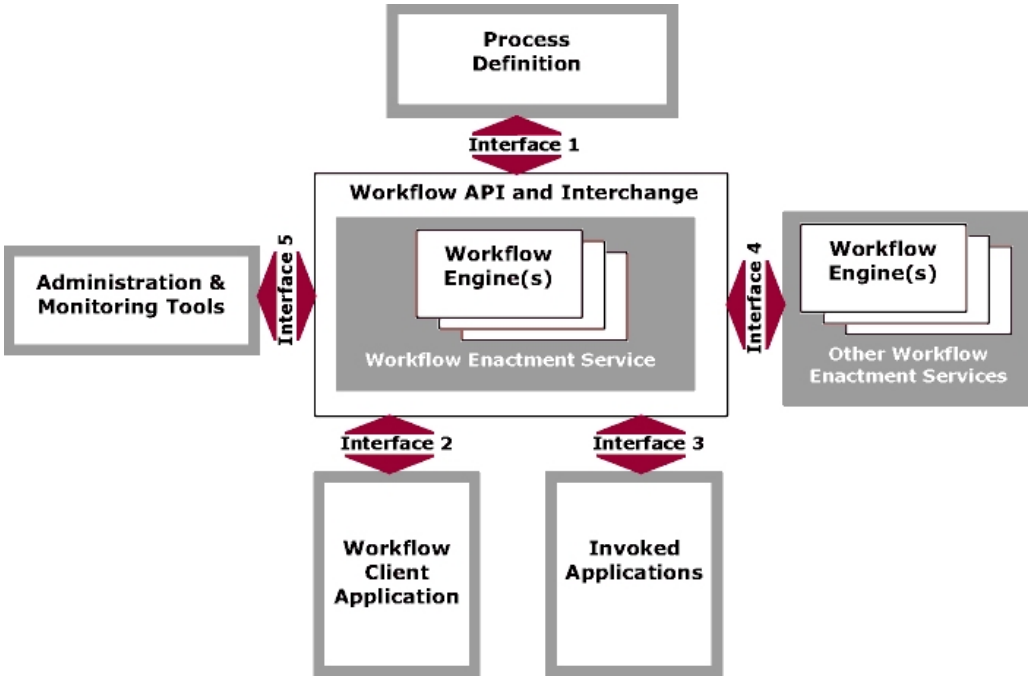


FIG. 7.6 – Architecture de référence du WfMC ©WfMC.

Ce modèle ne peut être appliqué tel quel mais il peut servir d'inspiration à différents niveaux dans la construction de l'architecture d'un système multi-agents. Chacun des composants qui figure dans ce modèle remplit une fonction bien précise. Si un composant ne se retrouve pas dans la nouvelle architecture combinant les deux technologies, les fonctions qu'il remplit devront par contre être toujours disponibles dans la plupart des cas :

- ***Définir les processus de workflow.*** Elle a lieu pendant la construction du système mais éventuellement aussi après, lorsque le système est déployé et actif. Il est donc nécessaire de pouvoir d'une part les spécifier lors de la conception du système et d'autre part, les modifier aisément lorsque le système est actif. Les diagrammes d'activité UML peuvent être utilisés par l'homme pour représenter les workflows. Ils permettent aux processus d'être communiqués, analysés et vérifiés au niveau de leur complétude et validité. Ces diagrammes fournissent des descriptions graphiques de haut-niveau des dépendances existantes entre les différentes tâches. Ils sont directement li-

sibles par l'homme, supportent l'expression de flux concurrents, l'utilisation de branchement conditionnel et la correspondance entre des activités et des acteurs spécifiques.

Malheureusement, aucun système ne peut les « digérer » automatiquement afin de les rendre actifs directement, ce n'est pas possible. Heureusement, il existe, comme nous l'avons déjà vu, des langages de description de processus tel que le BPEL4WS, un langage basé sur le XML. A partir des diagrammes d'activité, il est possible de développer un outil -il en existe déjà- qui pourrait automatiquement les traduire sous la forme de ce langage qui lui, peut être lu, compris et assimilé par le système multi-agents. Nous voyons déjà deux modifications à apporter suite à l'intégration des technologies du WM et MAS : au niveau méthodologique, insérer dans le processus de développement la définition des processus de workflow en utilisant les diagrammes d'activité UML et au niveau technique, disposer d'un outil permettant la traduction de ces processus de workflow en comportements agents au moyen par exemple du modèle BDI. Ajoutons que dans le cas de la plate-forme JACK que nous avons utilisée, les plans des agents peuvent déjà être dessinés graphiquement et que le formalisme utilisé est très proche de l'UML d'autant plus qu'à l'inverse de l'UML, il permet par exemple de prendre en charge des notions relatives aux agents comme celles du modèles BDI : objectifs, plans, événements, etc.

- ***Administrer et contrôler le système.*** A ce niveau, nous voyons la solution se dessiner rapidement puisque la plupart des plates-formes d'exécution de systèmes multi-agents offrent des possibilités de *monitoring*. Il est possible, au fil de l'exécution, de suivre le comportement des agents, de les tracer. Dans notre cas, suivre le comportement des agents signifie également suivre le comportement des processus de workflow puisque les agents collaborant ensemble représentent ni plus ni moins le système de workflow management.

- ***Servir d'interface avec un participant humain.*** Munis de leurs aptitudes mentales, les agents intelligents sont de plus en plus aptes à tenir un dialogue avec un utilisateur humain. Nul doute qu'ils pourront tout à fait satisfaire à cette tâche. Lorsque le système aura besoin de notifier un utilisateur de l'application, il pourra le faire de deux manières différentes, en faisant preuve de : *« push-communication »* ou *« pull-communication »*. Dans le premier cas, le système notifie le client activement, dans le second, c'est le client qui récupère dans le système la prochaine tâche qui lui incombe.

- ***Invoquer des applications externes sans intervention humaine.*** Comme nous l'avons déjà vu, les agents peuvent servir d'interface vers d'autres applications. Au lieu de définir des API pour l'interopérabilité entre applications, des messages émanant des agents peuvent être échangés. Les agents permettent également de s'interfacer facilement avec d'anciens systèmes (*le-*

gacy systems). Cela constitue un critère déterminant quand chacun sait que le succès d'un projet de workflow peut reposer sur l'intégration des applications existantes, spécifiques aux domaines et qui ont été développées auparavant, séparément de toutes applications de workflow.

– ***Servir d'interface vers d'autres systèmes de workflow.*** Comme pour le point précédent et au vu de la flexibilité de comportement et des possibilités d'adaptation et de communication des agents, cette tâche pourra aisément être prise en charge par ceux-ci.

Prendre en compte le fait que l'agent est l'acteur d'une société

Si une collection d'agents sociables, représentant des services individuels coopèrent et se coordonnent, ils vont permettre de mettre sur pied n'importe quel workflow qui est composé des services représentés. Les agents individuels, atteignent un comportement cohérent à travers des interactions coordonnées. Ils agissent dès lors comme une entité collective plus connue sous le nom de système multi-agents.

La mise sur pied d'un workflow par un système multi-agents peut être vue comme un acte de résolution de problème coopératif. *« La résolution de problème coopératif survient quand un groupe d'agents autonomes choisissent de travailler ensemble pour atteindre un objectif commun » [BV05].* Pour résoudre au mieux un problème coopératif, un agent dans une société multi-agents doit reconnaître que **le meilleur chemin pour atteindre un objectif est de faire appel aux autres agents, de leur demander de l'aide**.

Bien entendu, pour permettre l'existence de tel comportement de la part des agents, la méthodologie de développement d'un système multi-agents doit pouvoir en tenir compte, elle doit être adaptée en conséquence. Prenons le cas de Prometheus. Le fait de mettre sur pied une gestion du workflow devrait se retrouver dans les deux premières étapes de la méthodologie :

- ***La phase de spécification du système.*** Durant cette étape, les objectifs et les fonctionnalités principales du système sont mentionnés. De la même manière, il s'agira d'exprimer les objectifs et fonctionnalités du système de workflow que nous désirons mettre en place.

- ***La phase de design de l'architecture.*** Elle comprend, entre autre, la définition des types d'agents, un diagramme de vue d'ensemble du système, la définition du comportement du système par des protocoles d'interaction et une spécification des messages inter-agents. Nous avons déjà évoqué l'utilité des diagrammes d'activité UML pour représenter les processus de workflow. Ceux-ci trouveront certainement leur place durant cette étape. Nous insistons également sur l'importance des diagrammes et protocoles d'interaction qui devront permettre d'exprimer les rapports entre les agents et de spécifier comment ils devront collaborer à la résolution des problèmes coopératifs c'est-à-dire à l'atteinte des objectifs communs et la réalisation des processus de workflow.

IOP ou comment spécifier les agents et leurs comportements

Jusque là, nous avons avancé toute une série de modifications qu'il serait bon d'apporter à la fois aux méthodologies de développement et aux modèles d'architecture multi-agents pour que les premières permettent de mieux prendre en compte les caractéristiques des processus de workflow et pour que les seconds répondent aux besoins de ces systèmes. Dans cette optique, les auteurs Munindar P. Singh and Michael P. Huhns définissent dans leur article [SH99] la programmation orientée interaction ou IOP[12] comme une collection de techniques centrées autour de la notion d'interaction. Leur programme de recherche consiste à développer des primitives pour la spécification de systèmes d'agents et des contraintes sur leurs comportememts. Ces primitives sont étudiées aux niveaux des trois couches de l'IOP :

1. la coordination ;
2. l'engagement ;
3. la collaboration.

Dans leur recherche, ils se concentrent principalement sur la deuxième couche. Cela inclut des primitives telles que les sociétés, les rôles que les agents y jouent, les capacités dont ils ont besoin et les engagements qu'ils prennent ou ont besoin de la part d'autres agents. Dans la réalisation de ses engagements, l'agent peut créer une société, y opérer ou encore même la dissoudre, actions qu'il peut mener à bien seul, agissant de façon autonome. Dans ce contexte, la flexibilité est la caractéristique la plus importante que l'agent peut posséder. Nous laisserons le soin au lecteur de découvrir leurs travaux dans leur entièreté. Ce n'est en effet pas l'objet de ce point. Notre objectif est par contre de mettre en évidence le fait qu'il est nécessaire de posséder les moyens de bien spécifier les comportements des agents pour leur permettre de réaliser efficacement les processus de workflow. Les trois couches mentionnées ci-dessus sont une solution à ce problème. Mais nous ne prétendons pas qu'il s'agisse de la meilleure solution. Elle permet néanmoins de progresser dans la recherche d'un procédé de spécification des workflows à travers les agents.

[12] *Interaction-Oriented Programming*

Chapitre 8

Les agents orientés objectifs et le Workflow Management

Sommaire

8.1 Introduction

Au cours des chapitres précédents, nous avons démontré combien le domaine du workflow management peut bénéficier des technologies multi-agents. Bien que leur complémentarité ne fasse plus l'ombre d'un doute, nous avons montré qu'une définition de modèles adaptés, de méthodologies adéquates et d'architectures flexibles est encore nécessaire pour permettre des développements plus rapides, moins coûteux et de meilleure qualité. Nous savons qu'il existe différents types d'agents tels que les agents orientés événements (agents réactifs) et les agents orientés objectifs (agents proactifs). Au cours de ce chapitre, nous revenons sur les agents orientés objectifs que nous avons beaucoup utilisés lors de la réalisation de notre projet. Nous souhaitons mettre en évidence le fait qu'ils sont particulièrement bien adaptés à la gestion de processus de workflow management et qu'ils possèdent des qualités indéniables pour être performants dans ce domaine.

8.2 Trois catégories de processus de business

Dans son article *« Two and a Half Generations of Agent-Based Process Management »* [Deb], le professeur John Debenham de l'UTS[1] de Sydney distingue trois catégories de processus business :

1. ***les processus orientés activités*** : ils sont gérés par une architecture agent simple, unique et réactive ;
2. ***les processus orientés objectifs*** : ils sont gérés par un système multi-agents basé sur une architecture agent trois couches BDI ;
3. ***les processus orientés connaissances*** : ils sont gérés par le même système que les processus orientés objectifs à ceci près que le système est augmenté par une approche de gestion de la connaissance basée sur les types de tâches.

Les aspects automatisés d'un processus de business sont gérés par un système de management de processus qui applique une séquence d'activités pour chaque processus. Parmi les trois types de processus, seuls les processus orientés objectifs seront envisagés ici.

Selon L. Fisher [Fis], un processus business est *« un ensemble d'une ou plusieurs procédures ou activités liées qui réalisent collectivement un objectif du business, dans le contexte d'une structure organisationnelle définissant des rôles et des relations fonctionnels »*. Il est important de souligner que cette définition lie **un processus à un objectif**. Chaque processus, sous-processus ou activité est lié à un objectif.

Certains processus ont la propriété que leur terminaison survient lors de la réalisation de l'objectif du processus. D'autres possèdent la propriété que la décision concernant la réalisation ou non de l'objectif est ouverte au débat. Nous avons, entre autres, utilisé ce type de processus dans nos travaux. Etant donné l'objectif « des propositions de nouvelles matières doivent être souscrites au Conseil d'Accréditation de l'Université », le processus « tenir une réunion regroupant différents participants » peut être sélectionné pour atteindre cet objectif. Une telle réunion peut cesser après trois longues heures de discussion durant lesquelles des propositions seront faites. Après cette réunion, le processus « patron », qui pourrait être le Président du Conseil, peut alors décider si l'objectif a été atteint ou non. Cela implique que la terminaison d'un processus n'est pas toujours liée à l'atteinte de l'objectif de celui-ci.

8.3 Les processus orientés objectifs

Un processus orienté objectifs possède un objectif principal et peut être associé avec une séquence de sous-objectifs telle que la réalisation de cette séquence implique toujours la réalisation de l'objectif du processus principal. Chacun de ces sous-processus est associé avec au moins une activité et donc avec au moins

[1] University of Technology Sydney

une tâche. Certaines de ces tâches peuvent fonctionner mieux que d'autres mais il n'est pas possible, au départ, de savoir quelles tâches seront les plus performantes. Une tâche associée à une activité peut échouer sévèrement ou peut tout simplement être inefficace dans l'atteinte des objectifs de cette activité. En d'autres mots, **l'échec imprévisible d'une tâche est une caractéristique des processus orientés objectifs**. Si une tâche échoue, alors une autre façon d'atteindre l'objectif doit être trouvée.

Un processus orienté objectifs est semblable au processus orienté tâches (ou activités) dans le sens où pour chaque activité, chaque tâche est censée réaliser le sous-objectif de cette activité et est dissemblable en ce que la bonne exécution de ces tâches n'est pas prévisible. Ce qui fonctionne bien dans certains cas peut totalement échouer dans d'autres. De plus, la raison de cet échec peut se trouver en dehors du système. En conséquence, la gestion des processus orientés objectifs requiert à la fois une architecture logicielle qui peut faire face à l'échec et des techniques intelligentes de sélection de la meilleure tâche à réaliser dans la suite de la séquence, celle qui est la plus pertinente à exécuter pour atteindre le plus rapidement et efficacement les objectifs.

8.4 Architecture du système

Dans un système de gestion des processus orientés objectifs, un agent supporte un utilisateur. Ces agents gèrent le travail de leur utilisateur et gèrent le travail qu'un utilisateur a délégué à un autre utilisateur, et donc indirectement à un autre agent. La figure 8.1 nous montre les composants de chaque noeud dans ce système.

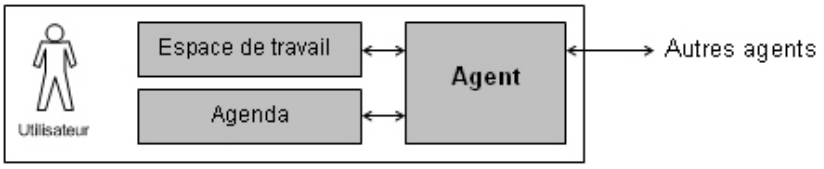


FIG. 8.1 – Un noeud du système dans une architecture multi-agents [Deb].

La délégation de sous-processus est le transfert de responsabilité pour un sous-processus d'un agent à un autre. Une stratégie de délégation décide à qui donner la responsabilité de réaliser telle ou telle tâche. Cette stratégie peut être très élémentaire mais peut aussi prendre la forme de systèmes complexes permettant d'effectuer des choix entre les trois principes suivants : maximiser la rentabilité (qualité du résultat par rapport aux efforts accomplis), maximiser les opportunités d'améliorer de pauvres performances et équilibrer la répartition des charges de travail.

La base du protocole d'interaction est la **coopération ouverte**. Par exemple, les agents sont sincères les uns envers les autres à propos du contenu de leur agenda. Dans la réalisation de notre projet, la sincérité des agents les uns envers les autres est primordiale. En effet, s'ils ne sont pas sincères, le système pourrait

devenir tout à fait inefficace ou pourrait favoriser certains agents et donc certaines personnes ce qui va évidemment à l'encontre de l'objectif de l'utilisation d'un tel système.

L'architecture conceptuelle des agents appartient à une classe d'agents, bien documentée, organisée en trois couches, à savoir l'architecture BDI que nous avons déjà étudiée au cours du chapitre 2. Nous n'allons pas revenir sur les concepts BDI que nous avons longuement évoqués auparavant. Par contre, ceci nous permet de faire le lien avec les agents sous-jacents au modèle BDI qui ne sont rien d'autre que les agents orientés objectifs. Ces agents sont capables de raisonner selon deux mécanismes différents : **réactif** et **proactif**. Le raisonnement proactif (ou *délibératif*) emploie la procédure non-déterministe suivante : « sur base de ses croyances actuelles, l'agent sélectionne les objectifs courants et pour chacun de ces objectifs, un plan pour l'atteindre et les prochaines tâches qu'il va réaliser (*intentions*) ». Le mécanisme de raisonnement réactif joue deux rôles : activer les actions d'échec si un plan échoue et lancer une procédure si son déclencheur[2] correspondant survient. Ce mécanisme est basé sur un principe de réaction à un événement spécifique.

8.5 Pertinence des agents orientés objectifs dans la gestion de workflow

Les agents orientés objectifs s'avèrent parfois complexes (par exemple, la bonne exécution d'un plan n'est pas toujours prévisible) mais font aussi preuve de qualités indéniables (ils peuvent par exemple fonctionner de manière délibérée) qui les rendent presque indispensables dans la réalisation d'un système multi-agents performant dans la gestion du workflow au sein d'une organisation. Nous sommes d'autant plus convaincus de leur utilité à la vue des éléments suivants :

- Dans le cas de processus orientés objectifs, il peut ne pas exister de manière de savoir quelle est la meilleure action à entreprendre lors de la prochaine étape. De plus, cette prochaine tâche peut impliquer de déléguer des responsabilités pour un sous-processus à un autre agent. Cette situation soulève deux problèmes : la ***sélection*** c'est-à-dire le fait de sélectionner le meilleur action pour un objectif donné et la ***délégation*** c'est-à-dire le fait de décider à qui demander de prendre la responsabilité de la réalisation de la prochaine tâche et par la suite, s'assurer que le travail est bien réalisé. Ces deux problèmes surviennent très régulièrement dans tout système de gestion de workflow et même tout simplement dans toute organisation en dehors de tout système automatisé. Face à cela, une solution possible est d'utiliser des agents orientés objectifs qui possèdent, en eux, la faculté de traiter ces deux problèmes. Comme nous l'avons déjà mentionné, une tentative de définir ce que signifie « meilleur (plan) » en termes de gestion de processus fonctionnel peut mener à des principes conflictuels comme : maximiser la rentabilité

[2] *Trigger* en anglais.

(qualité du résultat par rapport aux efforts accomplis), maximiser les opportunités d'améliorer des pauvres performances et équilibrer la répartition des charges de travail.

Pour réaliser la sélection, des statistiques à propos des performances de chaque plan sont rassemblées. Chaque agent lié à un utilisateur maintient des estimations pour les trois paramètres suivants : ***temps***, ***coût*** et ***probabilité de succès*** et ce pour tous les plans, sous-plans et activités du système. Le premier représente le temps total d'exécution d'un plan jusqu'à sa terminaison, le deuxième, le coût engendré par l'utilisation des ressources allouées à la réalisation de la tâche et le dernier, une valeur comprise entre 0 et 1 qui exprime la probabilité de succès de l'activité c'est-à-dire son exécution complète et correcte jusqu'à sa terminaison. Dans ce contexte, la probabilité de succès des plans (ou activités) suit une loi binomiale puisqu'il existe deux issues possibles : le succès ou l'échec.

De la même manière, la délégation s'opère au niveau des agents grâce à l'utilisation de deux paramètres. Chaque agent maintient un couple de valeurs pour chacun des autres agents du système : la ***qualité*** et la ***quantité*** relatives à la délégation de travaux d'un agent à chacun des autres pairs. Le premier caractérise la qualité du travail fourni précédemment par un agent et le second, la quantité de travail effectuée par ce dernier.

- Comme nous l'avons déjà expliqué dans le premier chapitre, les agents proactifs (orientés objectifs) sont capables de raisonnements intelligents. Dans le chapitre 2, nous avons vu un modèle de raisonnement d'agents : le modèle BDI. Ce dernier est particulier dans le sens où il offre une architecture qui supporte le développement d'agents intelligents qui raisonnent très similairement aux êtres humains. C'est dans ce sens que fonctionnent les agents orientés objectifs. Ils bénéficient actuellement des dernières recherches en matière d'intelligence artificielle distribuée et peuvent désormais simuler des comportements très proches des comportements humains. Dès lors, les agents proactifs sont à même de pouvoir prendre des décisions de la même manière que les humains. En conséquence, ils sont capables de supporter les workflows de manière beaucoup plus flexibles, ils sont plus robustes et peuvent jouer un rôle très important en tant qu'acteurs de ces workflows.

- Au cours du chapitre 7, sous-section 7.5.2 « Les apports supplémentaires des agents à prendre en compte lors de la spécification », nous avons mis en avant quelques qualités des agents. Nous avons également soulevé le problème d'autonomie dont souffrent les systèmes classiques de gestion de workflow, lacune qui se traduit par exemple, par l'inexistence d'un traitement des exceptions qui reste un des problèmes majeurs à résoudre. Désormais conscients des qualités indéniables que peuvent procurer les agents dans la construction de systèmes de workflow, nous désirons aller plus loin en affirmant que ces agents ne doivent pas être de n'importe quel type. Selon nous,

les agents orientés objectifs sont à l'heure actuelle les agents les plus adéquats qui puissent être utilisés dans des associations de technologies de type agent et workflow. Les agents orientés objectifs sont capables de prendre en charge une multitude d'opérations mais peuvent surtout résoudre le problème d'autonomie des applications de workflow. A même de modéliser des organisations (créant des mini-sociétés) en y interprétant des tâches « sans fin » (un agent joue le rôle d'un utilisateur), les agents orientés objectifs sont également en mesure de s'occuper du traitement des exceptions que nous avons évoqué ci-dessus et au cours du paragraphe 7.5.2.

Concrètement, des moyens sont mis à disposition des développeurs dans les plates-formes de développement de systèmes multi-agents. Nous avons déjà brièvement décrit au cours du chapitre 2 « Modèle BDI », au point 2.4 « Architecture BDI des agents JACK », le fonctionnement des agents développés sous Jack. Voyons cela encore plus en détail. Dans Jack, un plan décrit une séquence d'actions qu'un agent peut entreprendre quand un événement survient. A ce moment, plusieurs plans peuvent être appropriés. Pour décider quel(s) plan(s) est (sont) pertinent(s), chacun des plans intègre une méthode appelée *relevant()* qui, une fois exécutée, permet de savoir si le plan est adéquat à l'événement ou pas. Une fois les plans adéquats au type d'événement connus, l'agent détermine le(s) plan(s) applicable(s) en fonction du moment, de ses croyances et ce, grâce à la méthode *context()* qui peut par exemple interroger ses croyances atuelles au moyen d'expression(s) logique(s). Quand l'agent a terminé de déterminer l'ensemble des plans applicables, il en choisit un. Dans le cas d'agents orientés objectifs, l'agent peut initier une *procédure* de choix du plan le plus approprié (le choix ne sera pas arbitraire selon l'ordre des déclarations des plans mais l'agent effectuera des raisonnements de haut niveau[3]).

Lorsque l'agent exécute le premier plan, il commence par exécuter la méthode *body()*. Il s'agit d'une méthode un peu spéciale appelée méthode de raisonnement[4] qui diffère d'une méthode Java normale en incluant et utilisant des règles et conditions logiques. Chaque commande d'une méthode de raisonnement est considérée comme une instruction logique qui peut soit réussir, soit échouer. Jack fournit toute une panoplie d'instructions de raisonnement. Le plan est considéré comme réussi (son exécution), s'il s'exécute correctement jusqu'à sa terminaison. Cette explication (simplifiée) à propos des agents orientés objectifs nous permet de mieux comprendre comment ils fonctionnent. Ces moyens permettent de modéliser des comportements humains grâce à des exécutions de plans dont le succès dépend de chacune des instructions qui les composent. Il s'agit pour l'agent d'atteindre un objectif en s'assurant que toutes les attentes sont remplies et que tous les besoins sont comblés.

[3] En anglais, *meta-level reasoning*. Il s'agit de règles spécifiques qui permettent de déterminer un ordre de précédence dans l'exécution des plans.

[4] En anglais, *reasoning method*

« Les agents s'engagent au résultat désiré, pas au(x) méthode(s) choisie(s) pour l'atteindre » [Sof04]. Les agents orientés objectifs de Jack ont les moyens d'atteindre les objectifs de façon très complexe. Un agent orienté objectifs peut assembler plusieurs plans, appliquer des heuristiques pour le choix des plans ou encore agir intelligemment en cas d'echec d'un plan. Ils sont capables :

- *d'appliquer des raisonnements de haut niveau* - une technique pour permettre entre autre la sélection du plan le plus adéquat et approprié en fonction du type d'objectif et des croyances actuelles de l'agent ;
- *de reconsidérer des plans alternatifs en cas d'échec de plan* - si un plan échoue, cette technique permet à l'agent orienté objectifs de considérer d'autres plans pour atteindre ses désirs (objectifs) ;
- *de recalculer l'ensemble des plans applicables* - lors de l'échec d'un plan, l'agent peut assembler un nouvel ensemble de plans applicables qui exclut tout plan qui a échoué auparavant. L'agent orienté objectifs est donc capable de revoir ses intentions pour mieux s'adapter aux aléas du système.

Ces trois caractéristiques sont très importantes et permettent aux agents orientés objectifs de mieux répondre aux problèmes du traitement des exceptions (un événement inattendu survenant pendant l'exécution des processus de workflow) qui persistent encore dans les applications actuelles de workflow management.

- Au chapitre 6, sous-section 6.5.2 « Craintes », nous avons avancé les craintes et réserves, souvent légitimes, exprimées par les personnes redoutant les conséquences inconnues qui pourraient apparaître suite au déploiement massif de technologies de workflow management. Nous voulons montrer que l'utilisation de système multi-agents et en particulier d'agents orientés objectifs peut atténuer voire résoudre les problèmes suivants :
  - Peu de flexibilité : les systèmes multi-agents se montrent souvent efficaces dans de nombreux domaines. Contrairement à des applications classiques de gestion de workflow, des systèmes de gestion de workflow basés sur des agents et en particulier des agents orientés objectifs, peuvent s'adapter facilement à un environnement complexe et dynamique ;
  - Contrôle trop rigide : les agents orientés objectifs sont capables de réaliser une bonne partie du travail des utilisateurs. Ils peuvent prendre des décisions au nom de ces utilisateurs. Ces derniers se voient donc décharger d'un certain nombre de contraintes souvent imposées par des applications classiques de workflow telles que les ERP[5]. Ceci contribue au renforcement du sentiment de liberté des utilisateurs ;
  - Surveillance trop accrue : à nouveau, l'importance prise par les agents orientés objectifs dans le système permet d'atténuer le sentiment de contrôle

[5]ERP pour *Enterprise Ressource Planning*, SAP (`http://www.sap.com`) est le plus célèbre. Ces applications sont souvent critiquées par les employés qui estiment que leur liberté diminue et la pression augmente après la mise en place de tels systèmes.

et de surveillance dont pourraient souffrir les utilisateurs ;
- Trop peu de fonctionnalités : nous l'avons suffisamment montré auparavant, les systèmes multi-agents orientés objectifs permettent sans aucun doute de remplir un nombre impressionnant de fonctionnalités variées et de simuler des comportements et des expressions très divers.

8.6 Conclusion

Au cours des chapitres précédents, la qualité d'un workflow management, si celui-ci est basé sur un système multi-agents, a été mise en évidence. Au cours de ce chapitre, nous avons voulu aller plus loin. Ainsi, selon nous, en plus de baser une solution de gestion du workflow sur un système multi-agents, l'utilisation d'agents orientés objectifs peut également s'avérer très bénéfique à cette solution. Cette classe d'agents est associée à un nombre important de caractéristiques qui peuvent améliorer considérablement le workflow management au sein de l'organisation. Ces agents permettent notamment de simuler des comportements beaucoup plus proches de ceux des utilisateurs puisqu'ils fonctionnent de la même manière, sur base d'objectifs fixés. Nous sommes donc plus que convaincus de la pertinence de baser la gestion du workflow sur un système multi-agents orientés objectifs.

Quatrième partie

Etude de cas : Elaboration d'un système multi-agents

Chapitre 9

Etude de cas : Présentation

Cette étude de cas présente la conception et l'implémentation du système multi-agents que nous avons réalisé durant notre stage de quatre mois dans le département « Computer Systems » de la « University of Technology, Sydney ». Ce dernier a été supervisé par les professeurs S. Simoff et J.K. Debenham.

L'objectif du stage était la conception et l'implémentation d'un système multi-agents orientés objectifs supportant des processus de workflow s'opérant au sein de l'université. En l'occurrence, le professeur Debenham nous a proposé de modéliser un processus simplifié de création de cours dans lequel un directeur de département, un professeur, ou éventuellement une autre personne responsable de la création de nouveaux cours lance un appel à propositions de nouveaux cours.

Nous avons utilisé la méthodologie *Prometheus*, détaillée au chapitre 4 page 39, pour l'analyse et la conception du système multi-agents ainsi qu'un outil nommé ©Prometheus Design tool[1] qui offre un support pour la création des différents schémas propres aux deux premières phases d'élaboration de systèmes multi-agents.

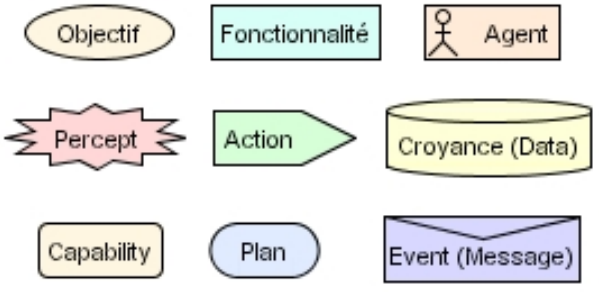


FIG. 9.1 – Légende des concepts de PDT

Lors la phase de développement, nous avons utilisé les logiciels JACK^{TM}[2] et Eclipse afin de développer respectivement les agents et leurs interfaces hommes-machines.

[1] Prometheus Design Tool (PDT) est disponible gratuitement sur `www.cs.rmit.edu.au/agents/prometheus`.

[2] Une version d'essai de JACK est disponible sur `www.agent-software.com`.

Notre application est composée de trois types d'agents. Le premier type d'agent, que nous avons appelé `DiaryAgent`, correspond à agenda ; le deuxième, appelé `TaskAgent` traite le processus de workflow. Enfin, le troisième type d'agent, `Name Server`, permet aux autres types de se localiser les uns les autres sur un réseau, et surtout de pouvoir connaître l'agent qui se cache derrière un utilisateur. Ce dernier est la pièce centrale du système, dans le sens où il doit être activé en permanence sur un ordinateur du réseau. Chacun des autres types d'agents est associé à un utilisateur. En pratique, chaque utilisateur dispose des deux types d'agents. Les deux agents propres à un même utilisateur peuvent interagir entre eux. De même, deux agents d'un même type appartenant à deux utilisateurs différents peuvent négocier entre eux, par exemple dans le cadre d'une planification de réunion pour le `DiaryAgent` ou dans le cadre du workflow pour le `TaskAgent`.

Nous avons développé un système de workflow basé agents (Chapitre 7. "Systèmes Multi-Agents et Workflow Management", paragraphe 7.4.2. "Un Workflow Management basé sur les Agents"). En effet, chaque agent est associé à un utilisateur. Cet agent est responsable de l'exécution d'une partie du workflow. Différents agents du même type interagissent entre eux et jouent chacun un rôle pour atteindre l'objectif du workflow.

Nous avons décomposé l'élaboration de notre projet en deux étapes. Dans un premier temps, nous avons conçu l'agent de type agenda électronique capable de (re)planifier des réunions de manière intelligente. La seconde étape consistait quant à elle à élaborer le deuxième type d'agent offrant une fonctionnalité de support de workflow management pour l'université.

Pour chaque étape du projet, nous avons fait correspondre une itération, chacune d'entre elle étant composée des phases d'*analyse*, de *design*, de *développement* et enfin d'une phase de *déploiement*.

L'analyse est la première phase de l'élaboration d'un SMA. La méthodologie Prometheus parle de phase de spécification du système, et consiste principalement en l'identification des objectifs, des fonctionnalités, des perceptions et des actions. Cette phase de spécification doit mettre en évidence les *objectifs* du système, ses *fonctionnalités* de base, les *scénarii* de ses opérations, et l'*interface* entre le système et son environnement.

La conception ou le design du système est la deuxième phase de l'élaboration d'un SMA. Prometheus distingue le *design de l'architecture* du *design détaillé*. Le design de l'architecture met en avant le couplage entre les fonctionnalités et les données par un schéma de «*Data Coupling*». Il doit également mettre en évidence les différents types d'agents qui interviennent dans le système multi-agents ainsi que leurs relations à l'aide du diagramme «*Agent acquaintance*», l'architecture du SMA sous la forme d'un «*System Overview Diagram*». Il est également d'usage de préciser les protocoles à l'aide de diagrammes d'interactions. Ces diagrammes sont réalisés à l'aide AUML[3], l'extension « Agent » d'UML. Le design détaillé définit précisément la structure et le fonctionnement de chaque agent.

[3] Agent Unified Modeling Language

Nous avons également intégré une analyse et une modélisation du workflow dans la seconde itération, à la section 11.2 page 131.

Les chapitres 10 et 11 détaillent les quatre étapes de l'élaboration de SMA pour chacune des deux itérations.

Chapitre 10

Première itération

Sommaire

10.1 Introduction

L'objectif de la première itération de l'élaboration de notre SMA est la création d'un agenda électronique capable d'enregistrer des évènements personnels d'un utilisateur et de planifier des réunions de groupe en tentant de trouver la meilleure plage horaire possible commune à tous les utilisateurs concernés par la réunion. Ce SMA est également capable d'annuler et de replanifier les réunions.

10.2 Première phase : L'analyse

Les objectifs du système

Nous avons identifié quatre objectifs principaux que nous présentons à la figure 10.1 : la (re)planification d'une réunion de groupe (`Plan ideal meeting` et `Replan meeting`), la planification d'un évènement personnel (`Plan personal event`), et l'interaction du système avec les utilisateurs (`User interaction`). Les objectifs principaux peuvent être décomposés en sous-objectifs. C'est le cas pour deux de nos objectifs. En effet, `Plan ideal meeting` et `Replan meeting` doivent être réalisés en tenant compte des disponibilités des utilisateurs et de leurs préférences. L'interaction avec l'utilisateur met en évidence deux sous-objectifs qui nous semblent importants. Il s'agit de mettre à jour l'agenda immédiatement après une planification pour éviter toute incohérence entre l'interface de l'utilisateur et

les croyances de l'agent `diary`. Il nous parait également nécessaire de notifier une réunion de groupe immédiatement après sa planification afin d'informer les invités de l'existence d'une nouvelle réunion dans l'agenda ou encore d'une réunion urgente, planifiée à la dernière minute.

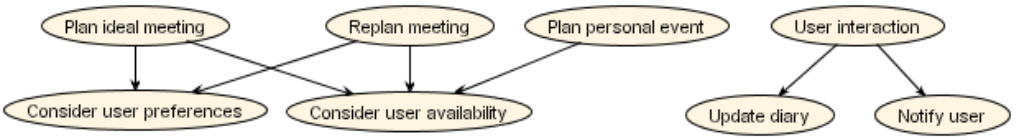


FIG. 10.1 – Objectifs du système

Les fonctionnalités

Comme nous le présentons à la figure 10.2, l'agent `Diary` compte cinq fonctionnalités. Sur cette figure, les fonctionnalités sont associées aux objectifs et aux actions qui les concernent.

La fonctionnalité `Meeting planning` permet à un utilisateur de planifier une réunion d'une certaine durée dans un certain délai avec les invités qu'il souhaite. Le système doit tenter de planifier cette réunion dans le délai voulu, en considérant les disponibilités et les préférences de tous les invités à la réunion. Dès lors, il effectuera l'action `Planning` qui consiste à planifier la réunion dans la tranche horaire qui satisfait le plus les invités. Il leur notifiera la réunion et mettra à jour les IHM des utilisateurs concernés. Il est cependant possible qu'aucune tranche horaire commune aux invités ne soit disponible pour la réunion. Dans ce cas, la planification échoue et son initiateur est prévenu et l'action `No planning` est prise pour signifier l'impossibilité de trouver une plage horaire commune.

`ReplanMeeting` est assez similaire. Cette fonctionnalité replanifie un réunion avant la deadline. Elle exécute l'action `Replanning`.

La fonctionnalité `Personal event planning` permet à l'utilisateur d'ajouter dans son agenda un évènement individuel, externe à l'entreprise (rendez-vous chez le médecin, etc.) ou tout simplement des jours de congé. L'action `Planning` est réalisée et l'agenda est immédiatement mis à jour.

Enfin, les fonctionnalités `Meeting cancellation` et `Personal event cancellation` permettent d'annuler un évènement qui était planifié dans l'agenda. Elles accomplissent l'action `unplanning`.

Les scénarii

Dans cette section, nous définissons les scénarii possibles des fonctionnalités que nous venons de définir. Un scénario Prometheus consiste en une série d'étapes. Chaque étape atteint un objectif (GOAL), reçoit une perception (PERCEPT), réalise une action (ACTION) ou exécute un autre scénario (SCENARIO). Il existe également un type d'étape OTHER pour couvrir les étapes inusuelles.

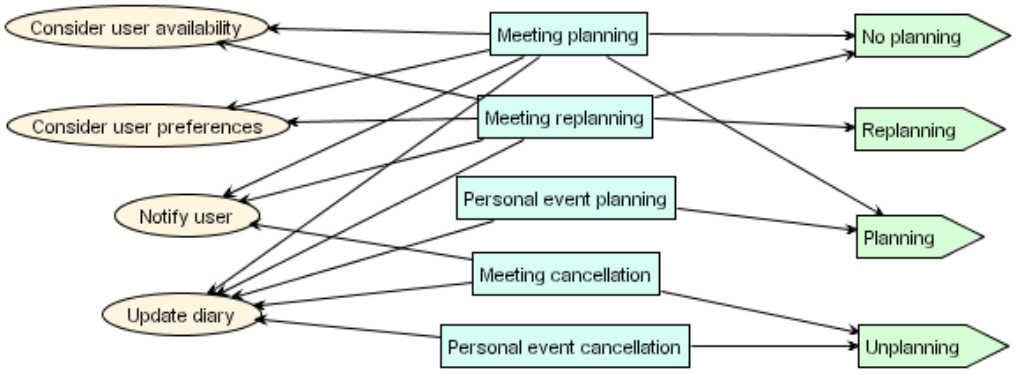


FIG. 10.2 – Fonctionnalités du système

SCÉNARIO PLAN PERSONALEVENT : Lorsqu'un utilisateur souhaite planifier un évènement personnel, l'agent reçoit une perception de l'environnement. Il vérifie dans l'agenda la disponibilité de l'utilisateur puis planifie cet évènement.

1. PERCEPT : Personal event request
2. GOAL : Consider user vailability
3. GOAL : Plan personal event
4. ACTION : Planning

SCÉNARIO CANCEL PERSONALEVENT : L'agent supprime directement un évènement personnel dès qu'un utilisateur le lui signifie.

1. PERCEPT : Personal event cancellation request
2. ACTION : Unplanning

SCÉNARIO PLANMEETING (ALTERNATIVE 1) : La fonctionnalité de planification est plus complexe. Elle peut se dérouler de trois façons différentes.

Dans la première alternative nous supposons que le déroulement de la planification se passe bien. Lorsqu'un utilisateur souhaite planifier une réunion, son agent reçoit une perception. L'agent contacte les invités à la réunion afin de considérer leurs préférences et leurs disponibilités. Ensuite, il sélectionne la tranche horaire qui convient le mieux à l'assemblée en calculant le produit des préférences de tous les invités pour chaque tranche horaire puis en sélectionnant celle qui correspond au produit maximal. Ensuite, l'agent initiateur de la réunion envoie une confirmation à tous les agents invités. Enfin, ces derniers planifient la réunion dans l'agenda de leur utilisateur avant de notifier la nouvelle réunion à ceux-ci via une interface homme-machine.

1. PERCEPT : New meeting request
2. GOAL : Consider users availability
3. GOAL : Consider users preferences
4. GOAL : Plan ideal meeting (select best slot)

5. Other : Confirm planning to invitees
6. Action : Planning
7. Goal : Notify users

Scénario PlanMeeting (alternative 2) : Dans ce cas, le processus de négociation est similaire à la première alternative, mais pendant la sélection de la tranche horaire idéale, une autre planification se déroule et sélectionne aussi cette tranche horaire chez un des utilisateurs invités. Il faut alors sélectionner la deuxième meilleure tranche horaire. S'il elle n'existe pas, la planification échoue et l'initiateur en est informé.

1. Percept : New meeting request
2. Goal : Consider users availability
3. Goal : Consider users preferences
4. Goal : Plan ideal meeting (select best slot)
5. Other : Best slot isn't free anymore for all users
6. Scenario : PlanMeeting

Scénario PlanMeeting (alternative 3) : Dans ce troisième cas, aucune tranche horaire n'est commune à tous les invités.

1. Percept : New meeting request
2. Goal : Consider users availability
3. Other : No common slot available
4. Action : No planning
5. Goal : Notify users

Scénario ReplanMeeting : Ce scénario détaille la replanification d'une réunion. Elle se déroule de façon similaire à une planification. Dès lors, nous ne présentons que l'alternative qui aboutit effectivement sur la replanification.

1. Percept : Replan Meeting Request
2. Goal : Consider users availability
3. Goal : Consider users preferences
4. Action : Replanning
5. Other : Confirm replanning to invitees
6. Goal : Notify users

Scénario CancelMeeting : Lorsque l'initiateur d'une réunion souhaite annuler celle-ci, son agent reçoit une perception. Il supprime la réunion de l'agenda et avertit les agents des invités à la réunion. Ces derniers suppriment également la réunion de leur agenda. Enfin, tous les invités sont avertis de l'annulation via l'interface homme-machine.

1. Percept : Cancel Meeting Request
2. Action : Unplanning
3. Other : Warn invitee from cancellation

L'interface du système

Cette dernière partie de l'analyse permet de relier les perceptions, les actions et les types de données avec un type d'agents. Le schéma 10.3 met en évidence les six perceptions et les quatre actions que nous venons d'identifier ainsi que les données (ou croyances) de l'agent. L'agent `diary` peut percevoir des requêtes de l'utilisateur pour une (re)planification ou une annulation de meeting ou d'évènement personnel. Nous pouvons également prendre connaissance des données utiles au bon fonctionnement de l'agent. Ainsi, l'agent `Diary` doit connaître les heures libres ou non, les réunions et les évènements planifiés, ses contacts et l'assistance aux réunions.

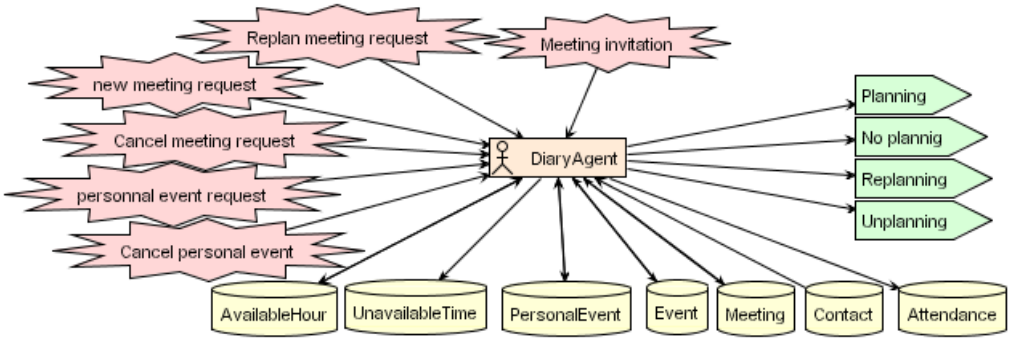


Fig. 10.3 – Interface du système avec son environnement

10.3 Deuxième phase : Le design

10.3.1 Design de l'architecture

Schéma de Data Coupling

Le schéma 10.4 définit les données dont le système doit disposer, ainsi que les relations entre ces données et les fonctionnalités. Les relations partant d'une fonctionnalité représentent une lecture tandis que celles arrivant sur une fonctionnalité représentent une écriture.

Les fonctionnalités nécessitent des connaissances sur les tranches horaires disponibles ou non et sur les évènements — qui peuvent être un évènement personnel ou une réunion — . Les fonctionnalités liées à la planification de réunion requièrent également une liste de contacts afin d'inviter des collègues à la réunion et de les prévenir en cas d'annulation ou de replanification d'une réunion. Elles nécessitent également la liste des invités aux réunions (attendance), ainsi que la liste des réunions. Les fonctionnalités liées à la planification d'évènement personnel

écrivent des évènements personnels. Une donnée `event` est soit un `personalEvent` soit un `meeting`.

Un schéma entité-association détaillant les croyances de l'agent se trouve dans l'annexe E page XXI.

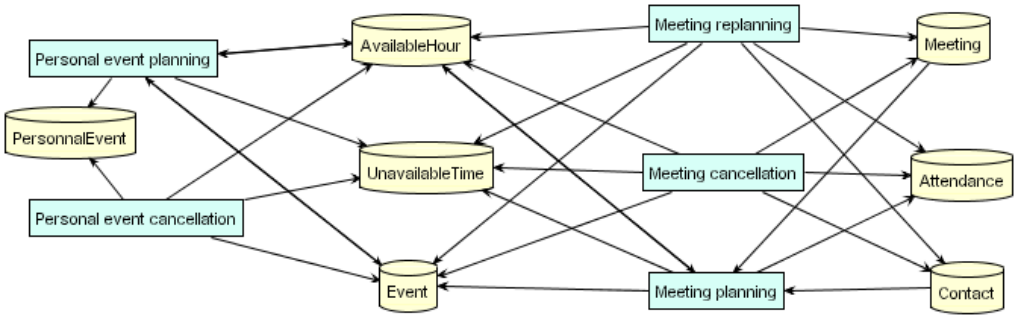


FIG. 10.4 – Couplage des données à l'intérieur du système

Diagramme d'Agent acquaintance

Ce diagramme permet de définir schématiquement les relations entre les agents. Lors de cette première itération nous réaliserons uniquement le type d'agent `diary`. Ce type schéma n'est donc pas pertinent pour le moment. Nous le retrouverons à la section 11.3.1 chapitre 11, page 137.

Le System Overview Diagram

Tous les composants de l'architecture de l'agent `diary` sont représentés sur la figure 10.5. Il comprend les perceptions, les actions et les données déjà détaillées à la section 10.3. Il met également en évidence les messages, appelés aussi évènements, que l'agent peut recevoir ou émettre, et grâce auxquels ils interagissent.

Par exemple, pour planifier une réunion, l'agent `diary` peut recevoir un évènement pour la planification d'une nouvelle réunion `NewMeetingEvent` provenant de l'utilisateur. Il est capable d'émettre des évènements d'invitation à des réunions `NewInvitationEvent` et en réponse recevoir des offres de tranches horaires libres `BidsEvent` des invités. Lorsque la meilleur plage horaire est sélectionnée, il peut l'envoyer sous la forme d'un évènement `ProposalEvent` et recevoir une réponse `ResponseEvent`. Une fois que les `ResponseEvent` des tous les invités sont reçus, l'agent initiateur de la réunion peut leur envoyer une confirmation de la date et de l'heure de la réunion.

L'évènement `CancelMeetingEvent` permet de prévenir les agents invités en cas d'annulation d'une réunion tandis que les autres évènements servent à la replanification de réunion.

Les diagrammes d'interactions

Les figures 10.6 à 10.10 représentent les interactions entre les agents et les utilisateurs ainsi que des interactions entre agents de même type. Chaque diagramme

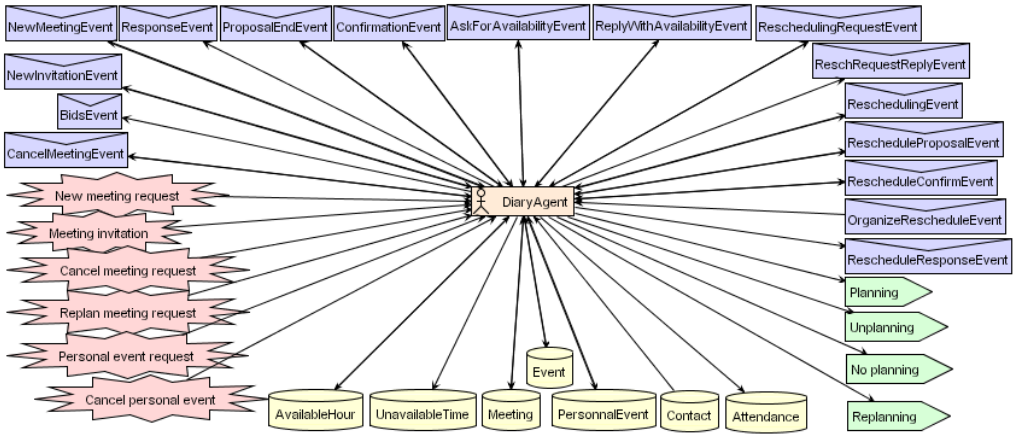


FIG. 10.5 – System Overview Diagram

d'interactions correspond à un scénario détaillé dans le section 10.2.

Ils sont réalisés sous la forme de diagrammes AUML dont la sémantique est détaillée dans le chapitre D à la page XVII.

Les fonctionnalités de planification et d'annulation d'évènement personnel ne présentent pas de comportement de négociation car elles ne concernent qu'un seul agent.

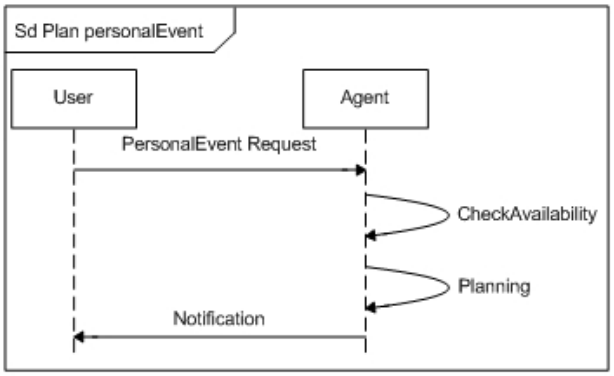


FIG. 10.6 – Diagramme d'interactions de la fonctionnalité de planification d'évènement personnel

Les fonctionnalités de (re)planification et d'annulation de réunion de groupe présentent un comportement de négociation assez complexe. Il est important de bien distinguer les rôles que l'agent `diary` peut prendre : rôle d'*initiateur* de réunion ou celui d'*invité* à une réunion.

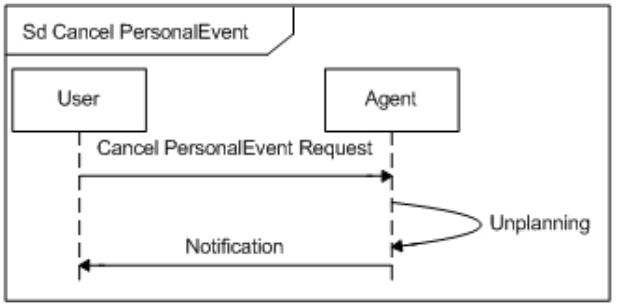


FIG. 10.7 – Diagramme d'interactions de la fonctionnalité d'annulation d'évènement personnel

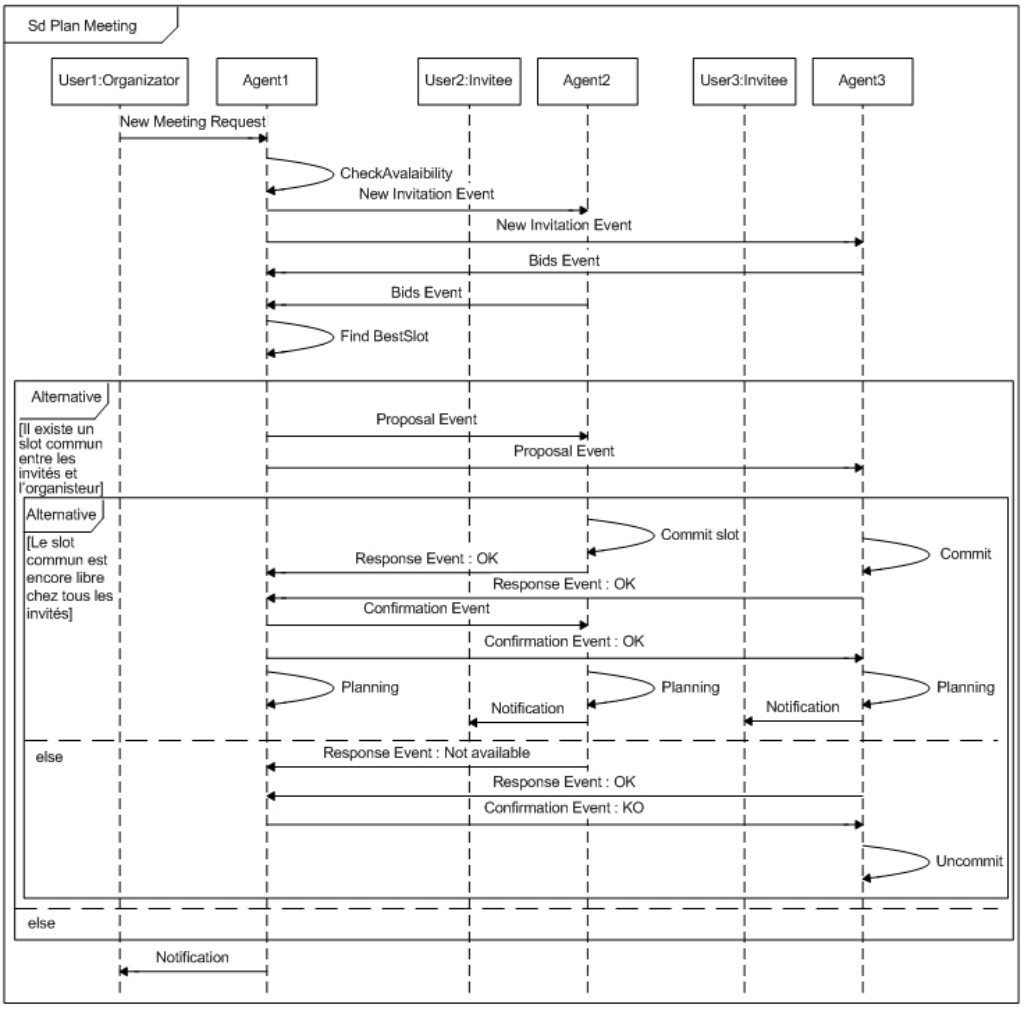


FIG. 10.8 – Diagramme d'interactions de la fonctionnalité de planification de réunion

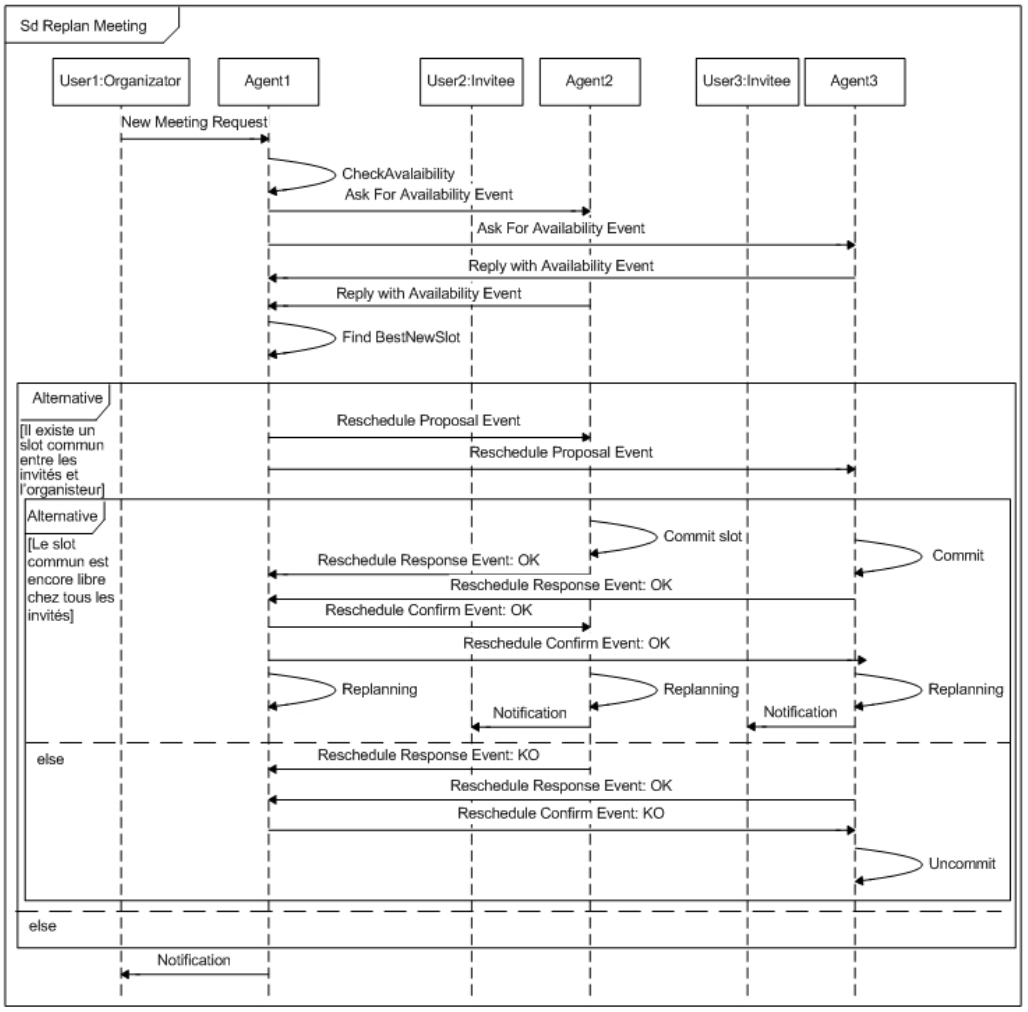


FIG. 10.9 – Diagramme d'interactions de la fonctionnalité de replanification de réunion

10.3.2 Design détaillé

Design détaillé de l'agent `Diary`

Le shéma 10.11 représente tous les éléments propres à l'agent `diary`. Il montre les relations entre les plans et les évènements, perceptions, actions et croyances, contrairement à la figure 10.5 — *System Overview* — qui ne présente pas encore les plans.

Le design détaillé présente une vue statique des composants de l'agent. Dès lors, il n'est pas aussi facile de distinguer le déroulement du processus que sur un diagramme d'interactions, mais ce schéma facilitera la création du squelette de l'agent sur la plate-forme JACK.

Dans la partie supérieure du schéma, nous pouvons distinguer les éléments de la fonctionnalité de planification de réunion. En plus des éléments dejà détaillés, il

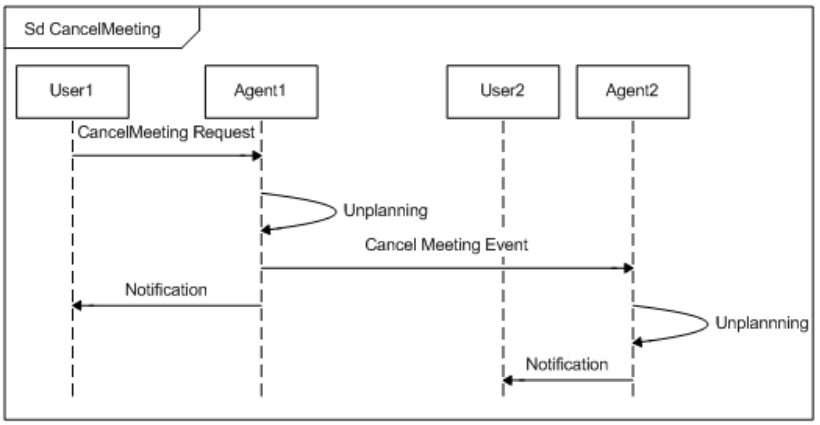


FIG. 10.10 – Diagramme d'interactions de la fonctionnalité d'annulation de réunion

contient des plans capables de mettre à jour les croyances, générer des évènements et réaliser des actions. Ils sont eux-mêmes déclenchés par des évènements. Ceux-ci interagissent de la manière dont nous l'avons détaillée dans le diagramme d'interactions (figure 10.8). A la figure 10.11, nous pouvons identifier les éléments de la fonctionnalité de planification de réunion, ceux de la fonctionnalité d'annulation, ceux de la fonctionnalité de replanification et enfin ceux des fonctionnalités liées aux évènements personnels.

Les fonctionnalités liées aux évènements personnels sont internes à un même agent. Il n'est pas nécessaire de créer des évènements. Un simple traitement, des mises à jour des croyances et une action sur l'environnement suffisent.

10.4 Troisième phase : Le développement

Nous avons utilisé la plate-forme JACK pour l'implémentation des agents ainsi qu'Eclipse pour l'implémentation des IHM. Comme nous l'avons déjà dit dans la section B.3 de l'annexe B, la méthodologie Prometheus a été conçue spécialement pour préparer le développement d'un système multi-agents avec JACK.

Le *diagram overview*, les diagrammes d'interactions et le design détaillé peuvent être dérivés aisément en SMA grâce à la plate-forme de développement JACK et tout particulièrement grâce à son outil graphique JDE. Pour cela, il faut dessiner le schéma F.2 situé en annexe page XXIV qui représente la synthèse de l'overview diagram 10.5 et du design détaillé 10.11. Ce schéma fait apparaître les relations entre un agent et ses messages, plans, perceptions et croyances. Il montre également celles entre les plans et ses messages et croyances.

Une fois ce diagramme dessiné à l'aide de JDE, la plate-forme JACK nous génère un squelette du SMA voulu, comprenant l'agent `diary`, des plans, des évènements et des croyances. Dès ce moment, il ne reste plus que la tâche qui consiste à compléter ce squelette, à l'aide de méthodes Java, d'instructions JAL[1]

[1] Jack Agent Language

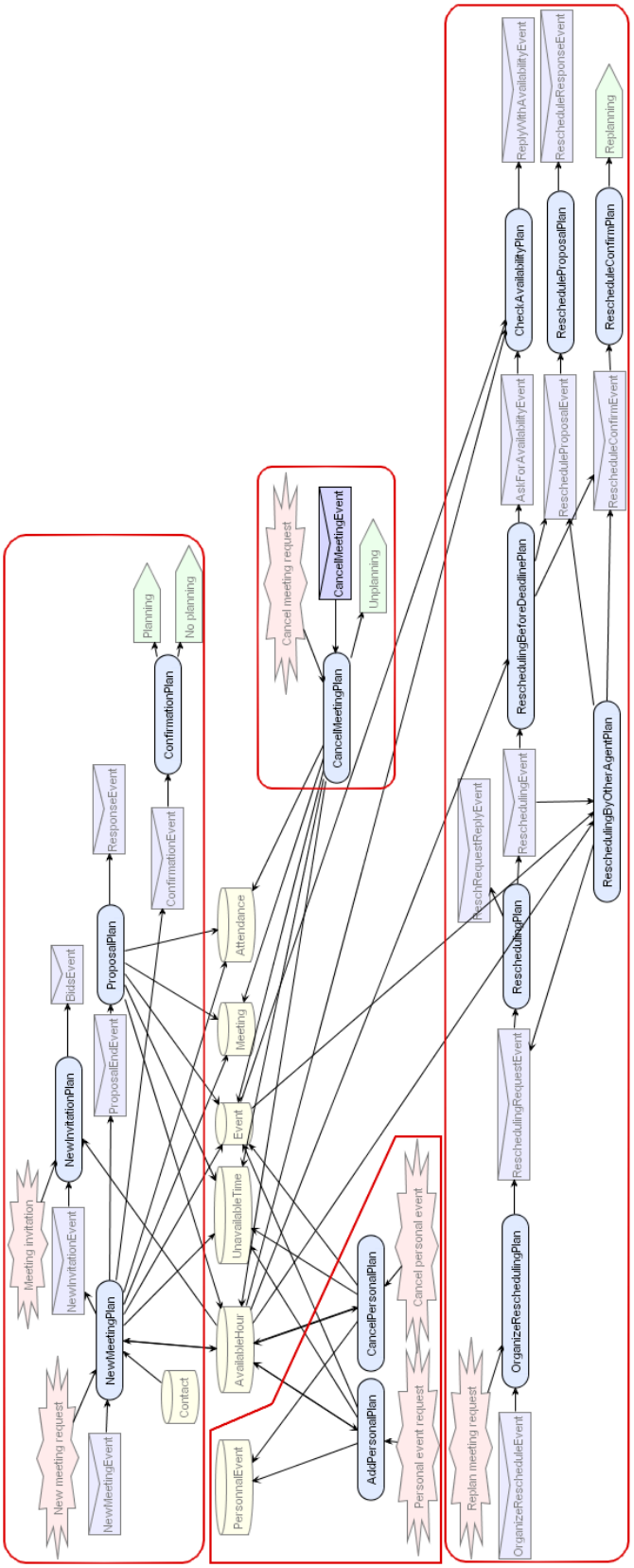


FIG. 10.11 – Design détaillé de l'agent `Diary`

et de requêtes sur les croyances.

10.5 Quatrième phase : Le déploiement

Le déploiement constitue la dernière phase de l'élaboration d'un SMA. Pour cette première itération, nous avons déployé cinq agents de type `diary` dans un environnement de tests.

Chapitre 11

Deuxième itération

Sommaire

11.1 Introduction

La deuxième itération consiste en l'élaboration d'un agent — que nous avons nommé `TaskAgent` — qui supporte la création de nouveaux cours à l'université.

Nous avons également élaboré un agent `NameServer` qui — comme son nom l'indique — est un serveur de noms. Il est capable de communiquer la localisation d'un agent donné du SMA à un autre agent qui en fait la demande. De cette manière, les agents n'ont plus besoin de connaître les adresses IP des agents qu'ils contactent. Le point G.2 de l'annexe G s'attarde sur les communications entre agents

11.2 Première phase : L'analyse

Durant notre stage à l'UTS, le professeur J.K. Debenham nous a proposé de modéliser la création de nouveaux cours. Ce processus se déroule de la manière suivante :

1. Une personne responsable du programme de cours constate que les cours de sa faculté sont trop peu nombreux et que certaines matières nouvelles ne sont pas proposées dans le programme ;
2. Cette personne est responsable de l'organisation d'une réunion du Conseil d'Accréditation. Cet organe se réunit pour trouver un certain nombre d'idées

de nouveaux cours. Son président communiquera les propositions à la personne responsable du programme, une fois la réunion terminée ;

3. Si le conseil d'accréditation n'est pas inspiré et que par conséquent, le nombre de nouveaux cours qu'il propose n'est pas satisfaisant, il est nécessaire de contacter des professeurs spécifiques qui transmettront à leur tour des propositions ;
4. Si le nombre de cours n'est toujours pas suffisant, il est possible de contacter des entreprises qui communiqueront leur idées en fonction de leurs besoins ;
5. L'étape suivante consiste en l'évaluation des coûts et des recettes des cours. Elle ne peut être entamée que si le nombre voulu de propositions de cours est obtenu ;
6. La dernière étape du processus est la sélection du (des) meilleur(s) cours proposé(s).

La figure 11.1 représente ce processus sous la forme de diagramme d'états-transitions UML où :

- le cercle plein représente l'état de départ ;
- le double cercle dont celui du centre est plein représente l'état final ;
- un rectangle représente un autre état ;
- une flèche représente une transition ;
- les transitions peuvent comporter des actions définies après un backslash (\) et certaines conditions définies entre des crochets ([...]).

Les objectifs du système

Les objectifs de cette itération sont directement dérivés du workflow que nous venons de définir. Nous pouvons identifier un nouvel objectif principal : la création d'un nouveau cours `Create new course`. Nous avons décomposé cet objectif en quatre sous-objectifs : La demande de proposition, la suggestion de nouveaux cours, l'évaluation des coûts des suggestions et la sélection des meilleurs cours.

Les fonctionnalités

Puisque le workflow est composé de trois grandes étapes, nous avons identifié trois fonctionnalités correspondantes :

- L'*appel pour des propositions de cours* — `Course appeal` — remplit le sous-objectif `Appeal for courses` et sa réponse `Suggest courses` ;
- La *demande d'évaluation des coûts* des propositions — `Cost appeal` — remplit le sous-objectif `Estimate cost of courses` ;
- La *sélection des meilleurs cours* — `Best course selection` — remplit le sous-objectif `Select the best course`.

Ensemble, ces trois fonctionnalités remplissent l'objectif principal.

Les scénarii

Dans cette section, nous définissons les scénarii relatifs aux nouvelles fonctionnalités que nous venons de définir.

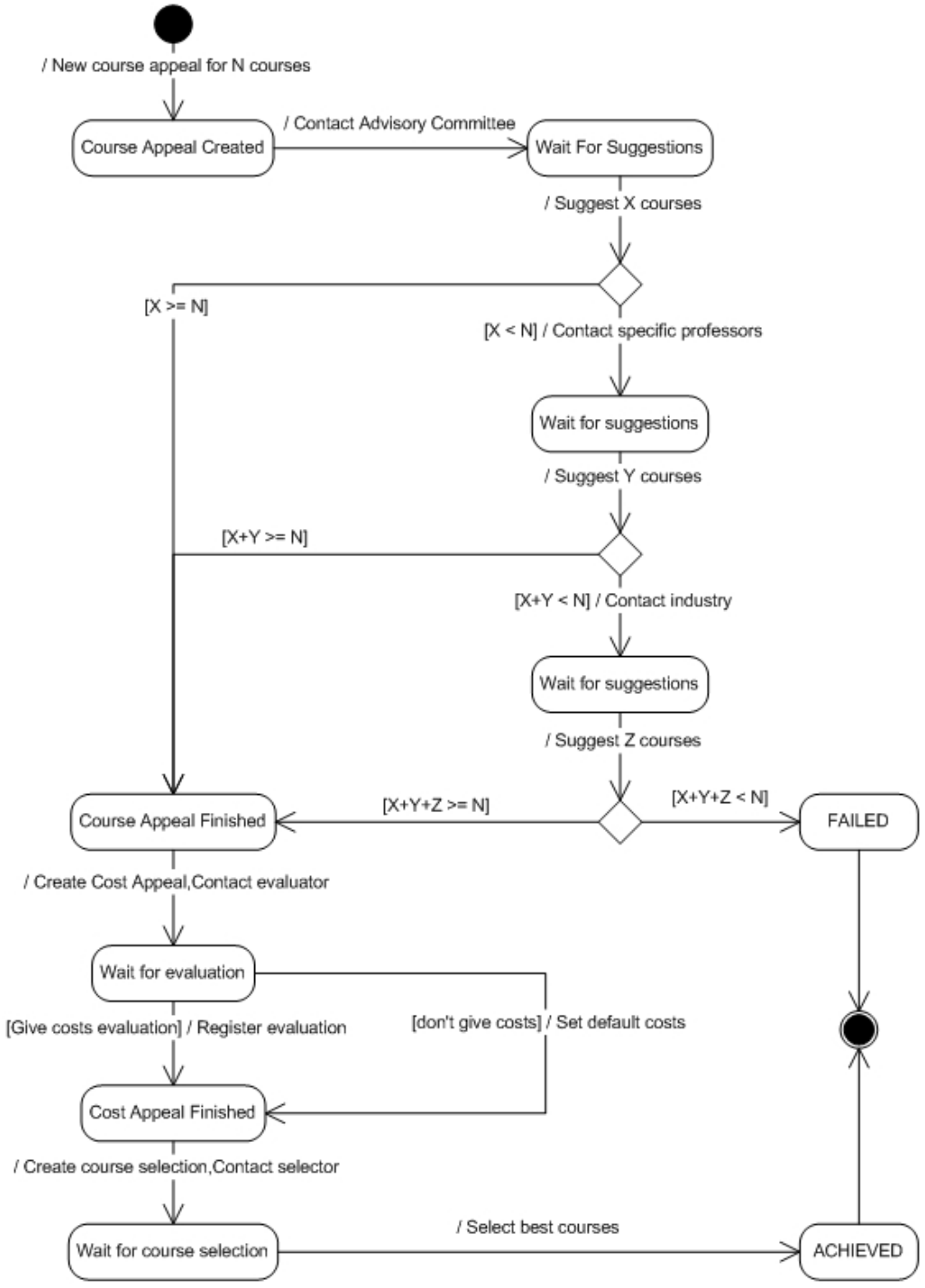


FIG. 11.1 – Modélisation du workflow avec un diagramme d'états-transitions

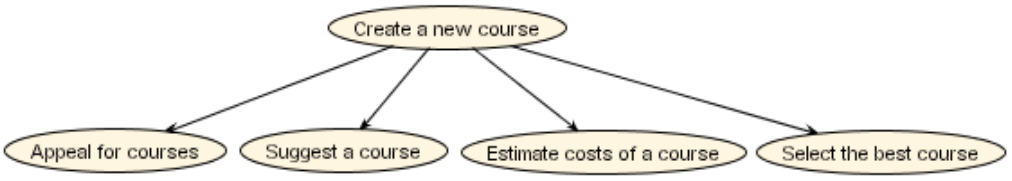


FIG. 11.2 – Nouveaux objectifs du SMA

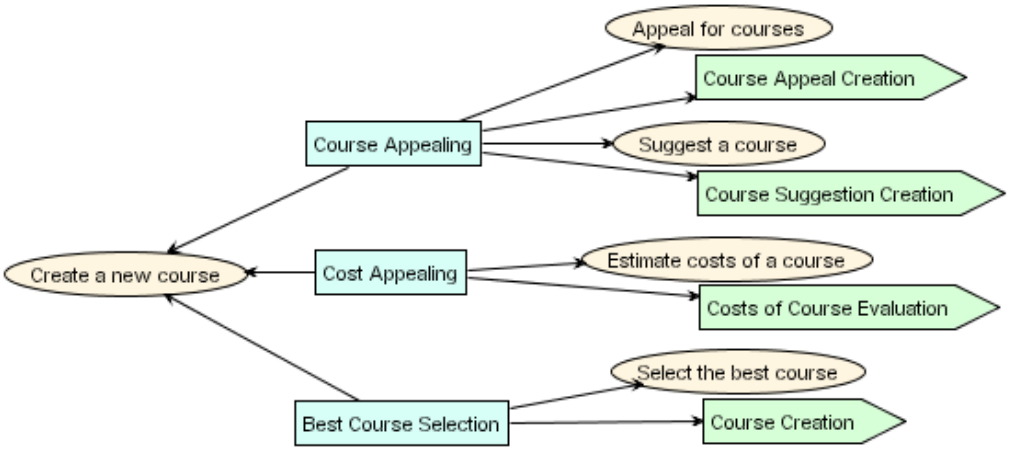


FIG. 11.3 – Nouvelles fonctionnalités du SMA

SCÉNARIO NEW COURSE : Ce scénario correspond au processus de workflow lié à la création d'un nouveau cours. Comme nous l'avons déjà mentionné dans la section *scénarii* au chapitre 10, un scénario peut être décomposé en plusieurs scénarii. Puisque le workflow que nous avons modélisé se déroule en trois temps, il nous paraît logique de décomposer ce scénario en trois scénarii.

1. SCENARIO : New course appeal scenario
2. SCENARIO : New cost appeal scenario
3. SCENARIO : Select best course scenario

SCÉNARIO NEW COURSE APPEAL (ALTERNATIVE 1) : La fonctionnalité d'appel pour des nouveaux cours peut se dérouler de trois façons différentes.

Un responsable souhaite créer un nouveau cours. Il émet un appel pour un certain nombre de nouveaux cours. Son agent reçoit une perception qui y correspond et propage l'appel au Conseil d'Accréditation. Celui-ci doit se réunir afin de mettre des idées en commun. Ensuite, le Président du Conseil devra envoyer les suggestions qui se sont dégagées de la réunion. Dans ce cas, nous considérons que le Conseil a suggéré assez de cours et que par conséquent l'appel pour de nouveaux cours est clos.

1. PERCEPT : New course appeal

2. Goal : Appeal for course
3. Action : Course appeal creation
4. Other : Contact advisory committee
5. Other : Wait for suggestion
6. Percept : New course suggestion
7. Action : Course suggestion creation

Scénario New course appeal (alternative 2) : Dans ce cas, nous supposons que le Conseil d'Accréditation n'a pas suggéré assez de cours. L'agent de l'initiateur du processus contacte alors des professeurs spécifiés par son utilisateur et attend leurs réponses. Si le nombre de suggestions du Conseil et des professeurs rencontre l'exigence de l'initiateur, l'appel pour des nouveaux cours se termine.

1. Percept : New course appeal
2. Goal : Appeal for course
3. Action : Course appeal creation
4. Other : Contact advisory committee
5. Other : Wait for suggestion
6. Percept : New course suggestion
7. Other : Contact specific professor
8. Other : Wait for suggestion
9. Percept : New course suggestion
10. Action : Course suggestion creation

Scénario New course appeal (alternative 3) : Dans ce dernier cas, le nombre de suggestions du Conseil d'Accréditation et des professeurs n'est pas suffisant, l'initiateur doit contacter des entreprises. Celles-ci répondent par un certain nombre de suggestions. Si le nombre total de suggestions n'est toujours pas suffisant, le processus échoue. Sinon, l'appel se termine.

1. Percept : New course appeal
2. Goal : Appeal for course
3. Action : Course appeal creation
4. Other : Contact advisory committee
5. Other : Wait for suggestion
6. Percept : New course suggestion
7. Other : Contact specific professor
8. Other : Wait for suggestion
9. Percept : New course suggestion
10. Other : Contact industry

11. Other : Wait for suggestion
12. Percept : New course suggestion
13. Action : Course suggestion creation

Scénario Cost appeal : L'évaluation des coûts d'un cours est la deuxième étape du workflow. L'initiateur du processus demande l'évaluation des coûts à la personne responsable. Lorsque celui-ci répond, le scénario se termine.

1. Percept : New cost appeal
2. Goal : Estimate cost of course
3. Other : Contact responsible for costs evaluation
4. Other : Wait for costs evaluation
5. Perpect : Cost evaluation
6. Action : Costs of course evaluation

Scénario Select best course : La sélection des meilleurs cours est la dernière étape du workflow. Lorsque l'évaluation des coûts est terminée, une personne désignée par l'initiateur du processus retient les meilleurs cours qui pourront être concrètement mis en place.

1. Percept : New best course selection
2. Goal : Select the best course
3. Action : Course creation

L'interface du système

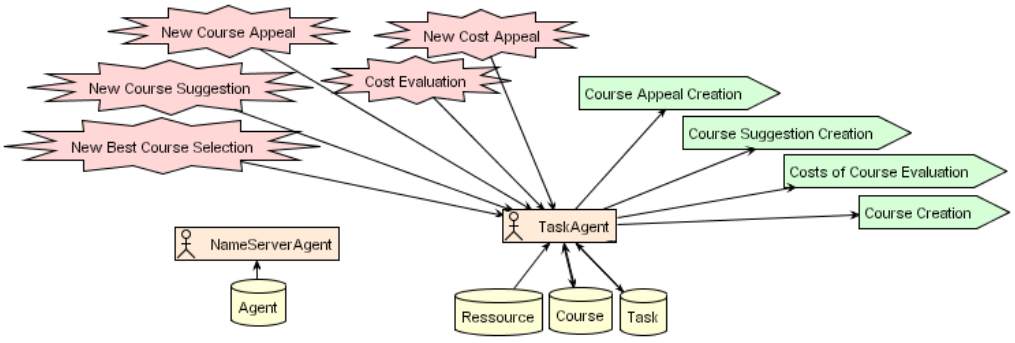


Fig. 11.4 – L'interface des nouveaux agents de notre système

Pour une meilleure lisibilité, ce schéma ne présente que l'interface des nouveaux agents `NameServer` et `Task` avec l'environnement. L'interface de l'agent `Diary` avec son environnement a déjà été détaillée au chapitre 10.

L'agent `NameServer` est lié aux données concernant la localisation des autres agents.

L'agent `Task` peut percevoir cinq types de perceptions (Demande de proposition de cours, suggestion de cours, demande de coûts d'un cours, évaluation des coûts et sélection d'un cours). Il est capable d'exécuter quatre actions (création du processus de demande de proposition de cours, création d'une suggestion de cours, création des coûts, création d'un cours), et possède trois types de connaissances respectivement sur les tâches à effectuer, sur les cours créés et sur les contacts (*ressources*).

11.3 Deuxième phase : Le design

11.3.1 Design de l'architecture

Schéma de Data Coupling

Les trois fonctionnalités de l'agent `Task` manipulent des croyances sur les tâches à effectuer et sur les cours proposés.

Un schéma entité-association détaillant les croyances des agents est joint dans l'annexe E page XXII.

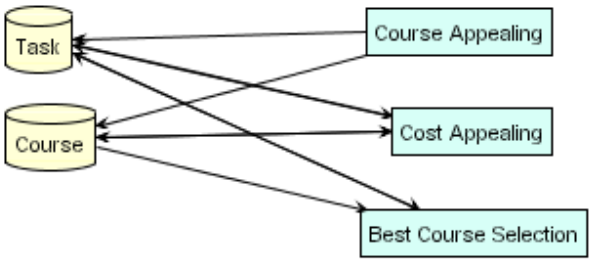


FIG. 11.5 – Croyances couplées aux nouvelles fonctionnalités

Diagramme d'Agent acquaintance

La figure 11.6 montre les relations entre les trois types d'agents. Les agents `Diary` et `Task` sont capables d'interagir avec l'agent `NameServer`. Ces deux premiers agents n'ont pas de relation directe entre eux.

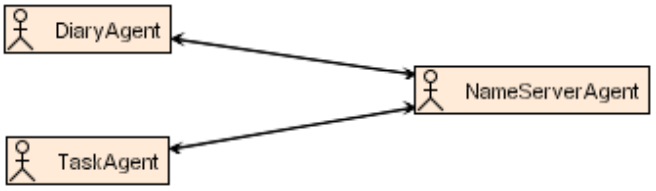


FIG. 11.6 – Relation entre les différents types d'agents

Le System Overview Diagram

La figure 11.7 présente les différents composants du SMA. Ce schéma est un peu simplifié. En effet, nous avons supprimé tous les éléments déjà détaillés dans le chapitre 10 afin de ne pas alourdir inutilement le schéma. Seuls les éléments nouveaux liés à l'agent `diary` sont représentés.

Les évènements `LocationRequest` et `LocationReply` permettent à un agent de connaître les adresses des autres agents du système, grâce à l'intervention de l'agent `NameServer`. Ce service est également disponible pour l'agent `Task`.

`NameServer` possède des croyances `Agents` sur la localisation des autres agents.

Nous remarquons également les perceptions et les actions liées à l'agent `Task` que nous avons déjà identifiés à la section 11.2 de ce chapitre.

Les évènements `CostAppealEvent`, `RequestForCostAppealEvent` et `EndOfCostSpecifEvent` permettent le bon fonctionnement de la fonctionnalité d'évaluation des coûts. `SelectCourseEvent` et `BestCourseEvent` sont liés à la fonctionnalité de sélection du meilleur cours. Les autres évènements se rattachent à la fonctionnalité d'appel pour des suggestions de nouveaux cours. Leur interaction est détaillées dans la sous-section suivante.

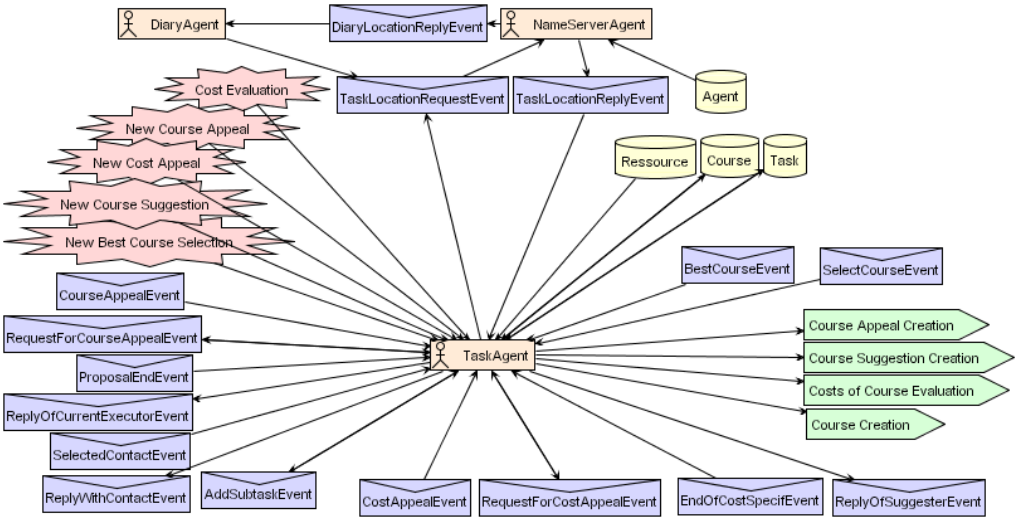


FIG. 11.7 – System Overview Diagram

Les diagrammes d'interactions

Dans cette section, Les figures 11.8 à 11.10 présentent les diagrammes d'interactions AUML des nouvelles fonctionnalités. Chaque diagramme correspond à un scénario des fonctionnalités décrites dans à la section 11.2.

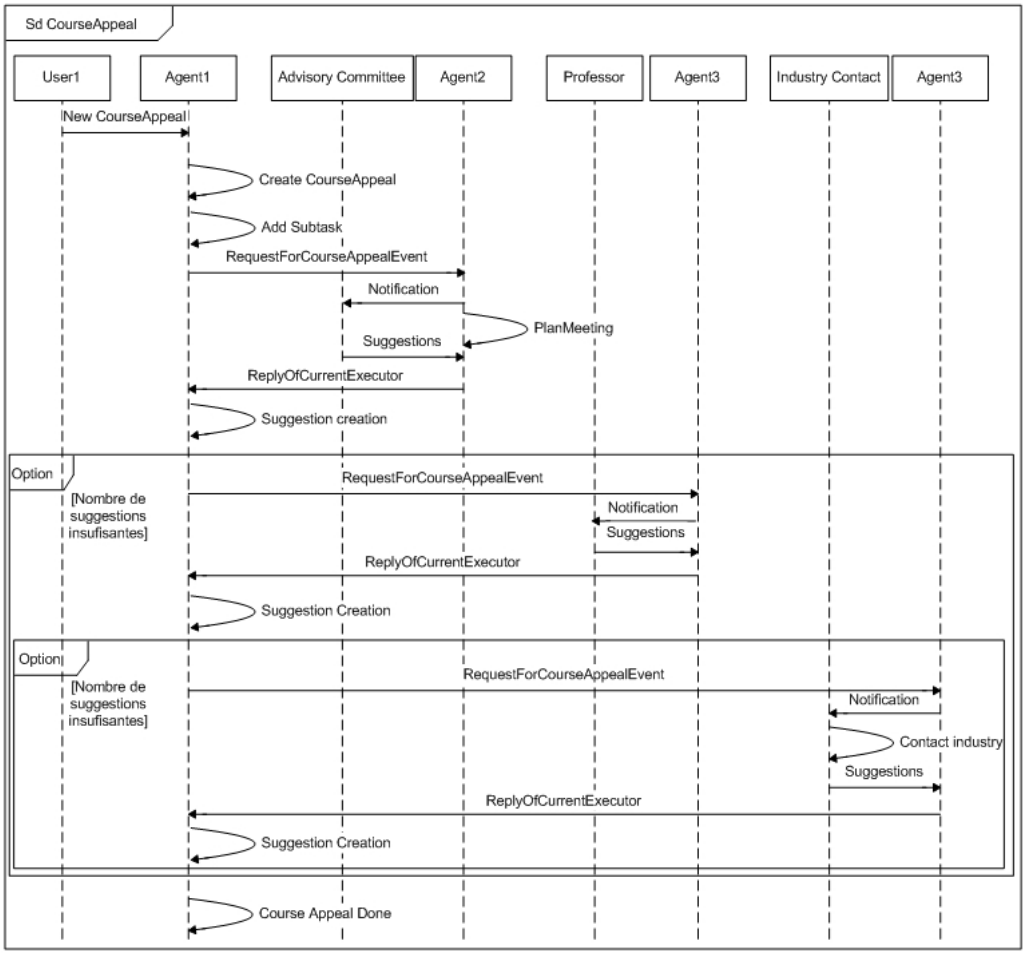


FIG. 11.8 – Diagramme d'interactions de la fonctionnalité `CourseAppeal`

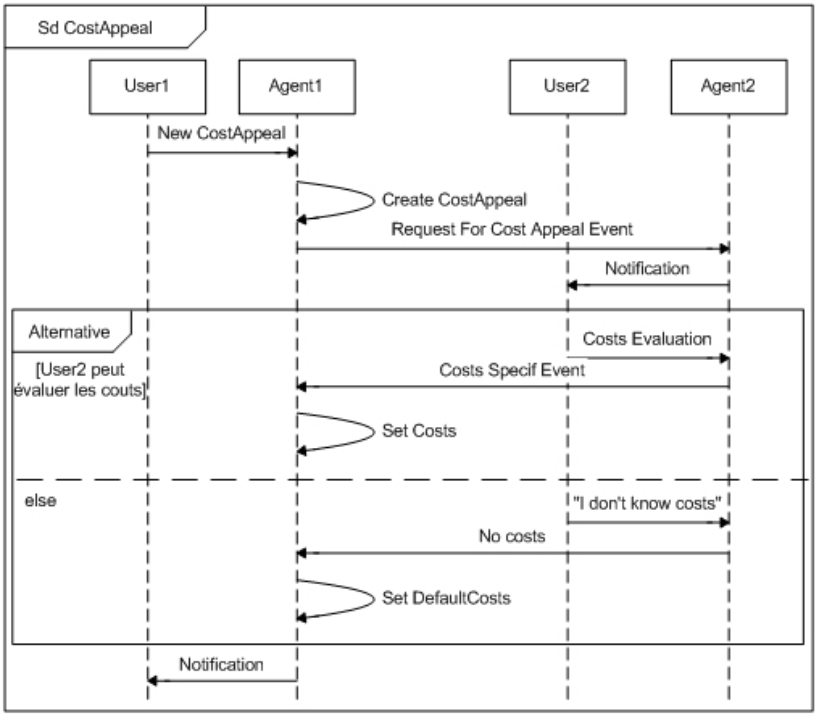


FIG. 11.9 – Diagramme d'interactions de la fonctionnalité `CostAppeal`

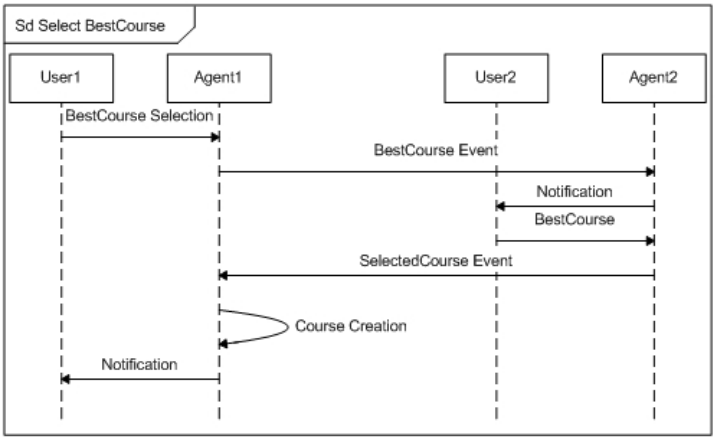


FIG. 11.10 – Diagramme d'interactions de la fonctionnalité `BestCourseSelection`

11.3.2 Design détaillé

Design détaillé de l'agent `NameServer`

La figure 11.11 met en évidence le processus de communication et le mode d'interaction entre l'agent `NameServer` et les autres agents. Des évènements de type `LocationRequest` peuvent être émis par les agents `Diary` et `Task`. Cet évènement est reçu par l'agent `NameServer`, qui déclenche immédiatement le plan `LocationRequest` et répond à la requête en produisant des évènements `Diary LocationReply` ou `TaskLocationReply` et qui sont respectivement captés par les agents `Diary` et `Task`.

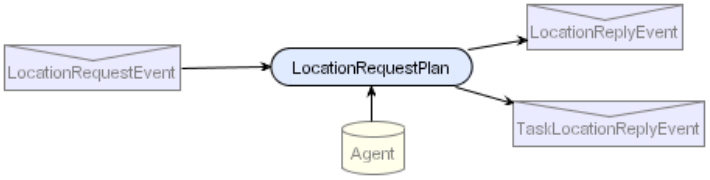


FIG. 11.11 – Design détaillé de l'agent `NameServer`

Design détaillé de l'agent `Diary`

Nous avons détaillé le design de cet agent durant la première itération. Le schéma se situe dans la section 10.3.2 à la page 129 .

Design détaillé de l'agent `Task`

La figure 11.12 est un peu complexe car elle présente tous les composants de l'agent `Task`. Nous avons disposé les éléments du schéma de manière à pouvoir identifier plus facilement les différentes fonctionnalités de cet agent.

La partie supérieure du schéma décrit la suite de perceptions, évènements, plans et actions liés à la création d'une nouvelle tâche de type `CourseAppeal`. En particulier, trois plans `AdvisoryCommittee`, `SpecificContact` et `Industry` correspondent aux trois étapes de l'appel pour des propositions de cours. Ces plans génèrent des évènements dans le but de contacter d'autres agents. Les réponses aux appels sont réalisées par le plan `ProposalEnd`. Celui-ci génère un évènement `ReplyOfCurrentExecutor` qui prévient l'agent initiateur de la tâche, de ses propositions de cours.

Les éléments liés à la création d'un nouvelle tâche de type `CostAppeal` sont décrits dans le milieu de l'image.

Enfin, la partie inférieure du schéma décrit les interactions entre les perceptions, évènements, plans et actions de la tâche `BestCourseSelection`.

11.4 Troisième phase : Le développement

A ce stade, nous pouvons dessiner les schémas dans JACK à l'aide de l'outil graphique JDE, générer les squelettes de code JAL et compléter ce squelette au

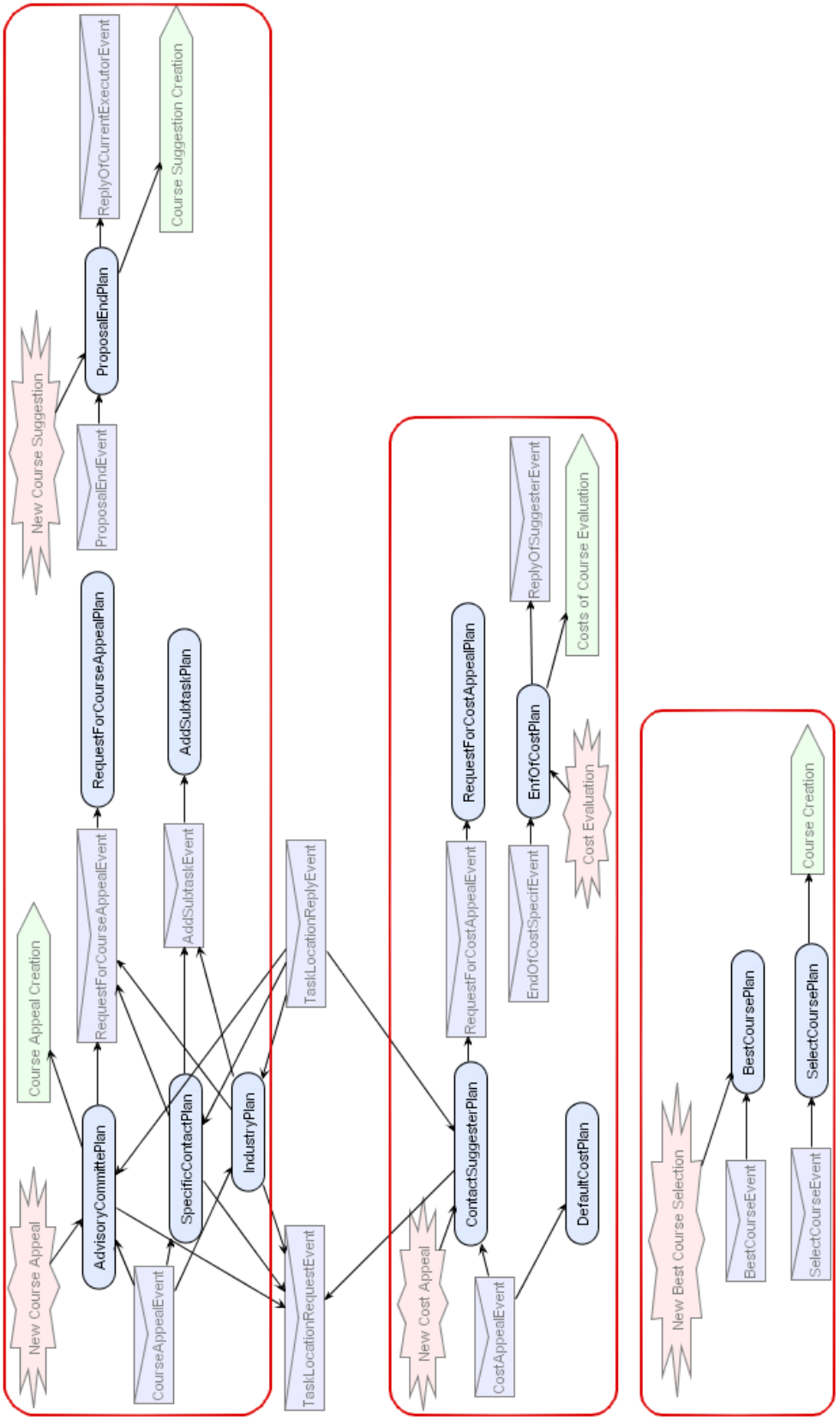


FIG. 11.12 – Design détaillé de l'agent `Task`

moyen de méthode Java, d'instructions JAL et de requêtes sur les croyances.

Ces schémas se trouvent à l'annexe F page F.3.

11.5 Quatrième phase : Le déploiement

Des explications concernant le déploiement et le lancement du système multi-agents sont exposées dans l'annexe G.

Conclusion

Dans ce mémoire, nous avons étudié les similarités entre les besoins de la gestion de workflow et les avantages que lui fournissent les systèmes multi-agents.

En pratique, nous avons développé une application multi-agents capable de gérer des workflows dans le cadre d'un processus de création de cours à l'université. Ce projet nous a permis de mettre en avant l'évidente complémentarité qui existe entre le workflow management et les systèmes multi-agents.

Nous sommes intimement convaincus que les systèmes multi-agents permettent d'aider considérablement les technologies de gestion de workflow à mieux faire face à leurs environnements en fournissant notamment une flexibilité accrue et un sentiment de contrôle atténué.

Les agents sont des entités autonomes, capables de comportements sociaux. Ils peuvent très facilement s'adapter à des environnements hétérogènes et bien qu'ils puissent se montrer concurrents, c'est surtout leur comportement coopératif qui est très intéressant. Travaillant ensemble, chacun des agents peut contribuer à la réalisation d'un objectif final. Les agents sont tout à fait capables d'implémenter des systèmes complexes alors qu'ils peuvent se montrer eux-mêmes très complexes.

Basé sur une unité centrale, les systèmes de gestion de workflow classiques manquent souvent d'automatisation, de réactivité, de gestion des ressources matérielles et humaines, de sémantique (la nature de l'information véhiculée n'a aucun intérêt pour le système), d'interfaces génériques vers d'autres applications (« legacy systems » par exemple) et d'interopérabilité entre les différents systèmes qui contribuent à la gestion des processus. Pendant que le workflow management se concentre sur la gestion de la logique des processus, cette technologie ne peut s'exprimer pleinement qu'en intégrant d'autres technologies pour contrôler complètement le processus de business. Ainsi les technologies liées aux agents fournissent fexibilité, réactivité et solutions intelligentes pour la gestion des processus de business. Une architecture de système distribuée devient envisageable, l'automatisation peut être accrue, les interactions avec d'anciens systèmes sont améliorées, la gestion des ressources est prise en charge et finalement l'interopérabilité entre des systèmes hétérogènes est rendue beaucoup plus simple.

Convaincus que des systèmes de workflow management basés sur des systèmes multi-agents constitue le présent et le futur du workflow, nous avons également avancé l'idée que des agents orientés objectifs sont les mieux adaptés au workflow management et qu'ils possèdent des qualités indéniables pour être performant dans ce domaine. Ils sont capables de sélectionner les meilleurs actions à entre-

prendre pour atteindre un objectif et de déléguer les tâches sous-jacentes à celui-ci. Ils peuvent simuler des comportements très proches des comportements humains comme la prise de décision notamment par l'utilisation de modèles de raisonnement tels que BDI. Ils peuvent modéliser des « mini-sociétés » en y interprétant des tâches sans fin (un agent joue le rôle d'un utilisateur) et rendent le système plus performant dans le traitement des exceptions qui peuvent survenir lors de l'exécution des processus de business.

Grâce à nos travaux, nous avons pu mettre en évidence la nécessité de méthodologies d'élaboration de systèmes multi-agents capables de supporter le workflow management.

Nous considérons que des travaux de recherche sont encore nécessaires. Une combinaison des technologies plus performante est possible. Nous estimons que certains besoins doivent être satisfaits et méritent d'être approfondis :

- une définition de méthodes de spécification de workflow qui prennent en compte la présence d'agents (étudier les possibilités offertes par des langages antérieurs tels que BPEL4WS, un langage de définition de processus) ;
- une définition de méthodes de spécification d'un ordre social initial et des relations inter-agents ;
- une définition de protocoles standards de communication avec des systèmes en place (« legacy systems »), avec d'autres types d'applications mais aussi surtout avec des utilisateurs ;
- une définition de primitives pour la spécification de systèmes d'agents et des contraintes sur leurs comportements (cfr. IOP page 104).

Finalement, une nouvelle méthodologie plus adaptée est nécessaire pour utiliser de façon optimale les spécifications de workflow et d'ordre social entre agents, les protocoles standards et les primitives pour la spécification de système d'agents de manière à supporter au mieux le développement de systèmes de gestion de workflow basés sur des agents proactifs. Ces besoins satisfaits, la combinaison de ces deux technologies n'en sera que plus efficace.

Bibliographie

[AGK+01] Josef Altmann, Franz Gruber, Ludwig Klug, Wolfgang Stockner, and Edgar Weippl. Using Mobile Agents in Real World : A Survey and Evaluation of Agent Platforms. In *2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS at Autonomous Agents*, Montreal, Canada, 29 May 2001.

[AR96] P. E. Agre and S. J. Rosenschein, editors. *Computational Theories of Interaction and Agency.* The MIT Press, Cambridge, MA, USA, 1996.

[AS94] Kenneth R. Abbott and Sunil K. Sarin. Experiences with workflow management : Issues for the next generation. In *CSCW*, pages 113–120, 1994.

[BP84] Giampio Bracchi and Barbara Pernici. The design requirements of office systems. *ACM Trans. Inf. Syst.*, 2(2) :151–170, 1984.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos : An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3) :203–236, 2004.

[Bra87] Michael Bratman. *Intention, plans, and practical reason.* Harvard University Press, 1987.

[Bra97] Jeffrey M. Bradshaw. *Software Agents.* AAAI Press, Mento Prak, USA, 1997.

[BRHL99] P. Busetta, R. Ronnquist, A. Hodgson, and A. Lucas. Jack intelligent agents - components for intelligent agents in java, 1999.

[BS95] Uwe M. Borghoff and Johann H. Schlichter. *Rechnergestützte Gruppenarbeit Ũ Eine Einführung in Verteilte Anwendungen.* Springer-Verlag Berlin Heidelberg, 1995.

[BV05] Paul Buhler and José M. Vidal. Towards adaptive workflow enactment using multiagent systems. *Information Technology and Management Journal*, 6(1) :61–87, 2005.

[CL90] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artif. Intell.*, 42(2-3) :213–261, 1990.

[CN99] Jaron Collis and Divine Ndumu. *The ZEUS Technical Manual.* British Telecommunications, BT Labs, 1999.

[Coa94] Workflow Management Coalition. The workflow management coalition specification, workflow reference model, November 1994. *Document Number TC00-1003.*

[Coa95] Workflow Management Coalition. The workflow reference model, 1995. Disponible à l'adresse `http://www.wfmc.org` (Dernière visite : 30 mai 2005).

[Deb] John Debenham. Two and a half generations of agent-based process management. (Not yet published). University of Technology, Sydney, `http://www-staff.socs.uts.edu.au/~`debenham/ (Dernière visite : 30 mai 2005).

[DG94] W. Deiters and V. Gruhn. The fun- soft net approach to software process management. *International Journal of Software Engineering and Knowledge Engineering*, 4(2) :229–256, 1994.

[DHW] Umeshwar Dayal, Eric Hanson, and Jennifer Widom. Active database systems, database system. edited by Won Kim, Addison Wesley Publishing Co. (New York : ACM Press, 1995).

[dKLW98a] Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dMARS. *Lecture Notes in Computer Science*, 1365 :155, 1998.

[dKLW98b] Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dmars. In *ATAL '97 : Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, pages 155–176. Springer-Verlag, 1998.

[DOBR00] Y. Demazeau, M. Occello, C. Baeijs, and P.-M. Ricordel. Systems development as societies of agents. In In Cuena et al editors, editor, *Knowledge Engineering and Agent Technology, In Series Frontiers in Artificial Intelligence and Applications*, volume 52. IOS Press, Amsterdam, The Netherlands, 2000.

[DW03] Khanh Hoa Dam and Michael Winikoff. Comparing agent-oriented methodologies. In *AOIS*, pages 78–93, 2003.

[DWS01] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3) :231–258, 2001.

[EGR91] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware : some issues and experiences. *Commun. ACM*, 34(1) :39–58, 1991.

[EN80] Clarence A. Ellis and Gary J. Nutt. Office information systems and computer science. *ACM Comput. Surv.*, 12(1) :27–60, 1980.

[Eng] Vincent Englebert. Chapitre 1. « requirements des systèmes distribués », pages 20 & 29. *Cours de Systèmes Distribués (FUNDP, Namur).* http ://www.info.fundp.ac.be/~ven/CIS/.

[ES89] E. A. Emerson and J. Srinivasan. Branching time temporal logic. In *Linear Time, Branching Time and Partial Order in Logics and*

Models for Concurrency., pages 123–172, Berlin - Heidelberg - New York, June 1989. Springer.

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, USA, 1994. ACM Press.

[FG96] Stan Franklin and Art Graesser. Is it an agent, or just a program ? : A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures and Language*, pages 21–35. Springer-Verlag, 1996.

[FG98] Jacques Ferber and Olivier Gutknecht. A meta-model for analysis and design of multi-agent systems". In *Proceedings of the 3rd International Conference on Multi-Agent Systems, (ICMAS'98)*, pages 155–176, August 1998.

[Fis] L. Fisher. Workflow handbook 2001. Workflow Management Coalition & Future Strategies, 2000.

[Fow97] M. Fowler. *Analysis Patterns : Reusable Object Models.* Addison-Wesley, 1997.

[GD02] Tony Garneau and Sylvain Delisle. Programmation orientée agent : évaluation comparative d'outils et environnements. In *Systèmes multi-agents et systèmes complexes (ingénierie, résolution de problèmes et simulation) – JFIADSMA'02*, Lille, France, 28 Octobre 2002.

[GH89] L. Gasser and M. N. Huhns. *Distributed Artificial Intelligence, Volume II.* Research Notes in Artificial Intelligence. Morgan Kaufmann publishers, San Mateo, CA, USA, 1989.

[GI88] M. Georgeff and F. F. Ingrand. Research on procedural reasoning systems. Technical Report Final report, phase 1 for NASA Ames research center, Moffet Field, CA, AI center, SRI International, October 1988.

[GPP+99] Mike Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Mike Wooldridge. The belief-desire-intention model of agency. In *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 1–10. Springer-Verlag : Heidelberg, Germany, 1999.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing : Concepts and Techniques.* Morgan Kaufmann, 1993.

[GR98] Michael P. Georgeff and Anand S. Rao. Rational software agents : From theory to practice. In *Agent Technology : Foundations, Applications, and Markets*, pages 139–159. Springer-Verlag, Heidelberg, Germany, 1998.

[GS86] Irene Greif and Sunil Sarin. Data sharing in group work. In *CSCW '86 : Proceedings of the 1986 ACM conference on Computer-supported*

cooperative work, pages 175–183, New York, NY, USA, 1986. ACM Press.

[Her03] A. Guerra Hernandez. *Apprentissage d'agents rationnels BDI dans un univers Multi-Agents.* PhD thesis, Université Paris 13, 2003.

[HL91] Keith Hales and Mandy Lavery. Workflow management software : The business opportunity. Technical report, Ovum Consultants, London, UK, 1991.

[HPW01] James Harland, Lin Padgham, and Michael Winikoff. Simplifying the development of intelligent agents. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AIŠ01)*, Adelaide, Australia, 2001.

[HRHL01] Nick Howden, Ralph Röonquist, Andrew Hodgson, and Andrew Lucas. Jack intelligent agents - summary of an agent infrastructure. In *5th International Conference on Autonomous Agents*, 2001.

[Huh87] M. N. Huhns. *Distributed Artificial Intelligence.* Pitman/Morgan Kaufmann, San Mateo, CA, USA, 1987.

[IBM00] IBM. Mqseries workflow. *White Paper*, 2000.

[InC00] InConcert. How is infostrada able to offer innovative, cutomized services to a rapidly growing customer base in real-time ?, 2000.

[JAD+] Stef Joosten, Gaston Aussems, Matthijs Duitshof, Richard Huffmeijer, and Erik Mulder. Wa-12 : an empirical study about the practice of workflow management. University of Twente, the Netherlands, July 1994.

[JB96] Stefan Jablonski and Christoph Bussler. *Workflow Management : Modeling Concepts, Architecture and Implementation.* International Thomson Computer Press, 1996.

[Jen01] Nicholas R. Jennings. An agent-based approach for building complex software systems. *CACM*, 44(4) :35–41, 2001.

[Joh88] Robert Johansen. *GroupWare : Computer Support for Business Teams.* The Free Press, New York, NY, USA, 1988.

[JW98] Nicholas R. Jennings and Michael J. Wooldridge. Applications of intelligent agents. In *Agent Technology : Foundations, Applications, and Markets*, pages '3–27. Springer-Verlag, Heidelberg, Germany, 1998.

[Kat90] Robert L. Katz. Business/enterprise modeling. *IBM Systems Journal*, 29(4) :509–525, 1990.

[KR03] James Kurose and Keith Ross. *Analyse structurée des réseaux : Des applications de l'internet aux infrastructures de télécommunication, 2ème édition.* Pearson Education France, 2003.

[LMP03] M. Luck, P. McBurney, and C. Preist. *Agent Technology : Enabling Next Generation Computing (A Roadmap for Agent Based Computing).* AgentLink, 2003.

[Mae91] Pattie Maes. *Designing Autonomous Agents : Theory and Practice from Biology to Engineering and Back*. MIT Press, Cambridge, MA, USA, 1991.

[MB92] David Marca and Geoffrey Bock. *Groupware : Software for Computer-Supported Cooperative Work*. IEEE Computer Press, 1992.

[MC94] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1) :87–119, 1994.

[OBDK00] M. Occello, C. Baeijs, Y. Demazeau, and J-L. Koning. Mask : An aeio toolbox to develop multi-agent systems. In In Cuena et al editors, editor, *Knowledge Engineering and Agent Technology, In Series Frontiers in Artificial Intelligence and Applications*, volume 52. IOS Press, Amsterdam, The Netherlands, 2000.

[Ode02] James Odell. Objects and agents compared. *Journal of Object Technology*, 1(1) :41–53, 2002.

[Opp88] S. Opper. A groupware toolbox. 13(13) :275–282, December 1988.

[OW98] P.D. O'Brien and M.E. Wiegand. Agent based process management : applying intelligent agents to workflow. *the Knowledge Engineering Review*, 13/2 :161–174, 1998.

[PD97] Parunak and Van Dyke. Go to the ant : Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75(0) :69–101, 1997.

[PW02] Lin Padgham and Michael Winikoff. Prometheus : A methodology for developing intelligent agents. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *AOSE*, volume 2585 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2002.

[RD00] Pierre-Michel Ricordel and Yves Demazeau. From analysis to deployment : A multi-agent platform survey. In *ESAW '00 : Proceedings of the First International Workshop on Engineering Societies in the Agent World*, pages 93–105. Springer-Verlag, 2000.

[Rei94] Berthold Reinwald. Workflow management. In *13th IFIP World Computer Congress*, Hamburgo, Alemanha, August 1994.

[RG91] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc. : San Mateo, CA, USA, 1991.

[RG95] Anand S. Rao and Michael P. Georgeff. BDI-agents : from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, USA, 1995.

[SH99] Munindar P. Singh and Michael P. Huhns. Multiagent systems for workflow. *International Journal of Intelligent Systems in Accounting, Finance and Management*, June 1999.

[Sho91] Y. Shoham. Agent-o : a simple agent language and its interpreter. In *Proceedings of the Ninth Conference on Artificial Intelligence*, volume II, pages 704–709, Anaheim, California, 1991. MIT Press.

[Sof04] Agent Oriented Software. *JACK Intelligent Agents - Agent Manual.* Agent Oriented Software Pty. Ltd., Carlton South, Victoria, 3053, Australia, May 2004.

[Tho93] Sarah Rebecca Thomas. *PLACA, an agent oriented programming language.* PhD thesis, 1993.

[Tra96] K. Trammel. Workflow without fear. *Byte*, April 1996.

[Vrb03] Pavel Vrba. Java-based agent platform evaluation. In *HoloMAS*, pages 47–58, 2003.

[WGHS99] Mathias Weske, Thomas Goesmann, Roland Holten, and Rüdiger Striemer. A reference model for workflow application development processes. In *WACC '99 : Proceedings of the international joint conference on Work activities coordination and collaboration*, pages 1–10, New York, NY, USA, 1999. ACM Press.

[Wil91] Paul Wilson. *Computer Supported Cooperative Work : an introduction.* Oxford, England Norwell, MA, Intellect ; Sold and distributed in the U.S.A. and Canada by Kluwer Academic Publishers, November 1991.

[WJ95] Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents : Theory and practice. *Knowledge Engineering Review*, 10(2) :115–152, 1995.

[WJK00] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3) :285–312, 2000.

[Woo02] Michael J. Wooldridge. *An Introduction to Multiagent Systems.* Wiley, Chichester, UK, 2002.

[WP04] Michael Winikoff and Lin Padgham. *Developing Intelligent Agent Systems : A Practical Guide.* Halsted Press, 2004.

[YMS01] Yuhong Yan, Zakaria Maamar, and Weiming Shen. Integration of workflow and agent technology for business process management. In *The Sixth International Conference on CSCW in Design*, London, Ontario, Canada, July 12-14 2001.

Annexes

Annexe A

Formalisation du modèle BDI

Depuis la théorie de Bratman [Bra87] de nombreux travaux ont été menés pour formaliser et implémenter le modèle BDI, notamment par Cohen et Levesque [CL90]. Dans leur formalisme, les intentions sont définies en terme de séquences temporelles par rapport aux objectifs et aux buts d'un agent.

Dans cette annexe, nous présenterons un formalisme alternatif des mondes possibles que nous devons à Rao et Georgeff [RG91]. Trois aspects importants de ce formalisme méritent notre attention. En premier lieu, les intentions sont traitées comme une classe à part, sur un même pied d'égalité qu'une croyance ou qu'un objectif, contrairement au modèle de Cohen et Levesque. Deuxièmement, le formalisme fait la distinction entre les choix d'actions que réalisent les agents, et les possibilités d'issues d'une action. Dans le premier cas, l'agent peut choisir entre différentes issues, dans le second cas, c'est l'environnement qui fait le tri. Troisièmement, ce formalisme spécifie l'interrelation entre les croyances, les objectifs et les intentions.

Dans les sections suivantes, nous détaillerons la syntaxe, la sémantique et les axiomes de ce formalisme. Il est important de préciser que les travaux de Rao et Georgeff présentent ce formalisme d'une manière plus détaillée. Ils fournissent entre autres des théorèmes et des preuves de leur modèle qui dépassent de loin le cadre de ce travail, et de l'implémentation du raisonnement BDI de nos agents.

A.1 Description syntaxique

Le formalisme utilisé pour décrire les structures du modèle BDI intègre également des formules de logique du premier ordre. Il est étendu par une logique propositionnelle temporelle similaire au formalisme *Computation Tree Logic* (CTL) [ES89].

Ce formalisme utilise deux types de formules : les *formules d'état* et les *formules de chemin*. Il inclut des opérateurs modaux et temporels. Les opérateurs modaux sont *optional* et *inevitable* et opèrent sur les formules de chemin. Les opérateurs standards temporels $\bigcirc$ (suivant), $\Diamond$ (finalement), $\Box$ (toujours), $\bigcup$ (jusqu'à) opèrent sur les formules d'état et de chemin.

Soit :

- ϕ, ϕ_1 et ϕ_2 des formules d'états ;
- ψ, ψ_1 et ψ_2 des formules de chemin ;
- x une variable ou un évènement ;
- e un évènement.

Les formules ci-dessous sont des *formules d'état* :

- toute formule du premier ordre ;
- $\neg\phi_1, \phi_1 \vee \phi_2$, et $\exists x\ \phi_1(x)$;
- $succeeds(e), fails(e), does(e), succeeded(e), failed(e)$ et $done(e)$;
- BEL(ϕ), GOAL(ϕ), et INTEND(ϕ)
- $optional(e)$

Les formules suivantes sont des *formules de chemin* :

- toute formule d'état ;
- $\neg\psi_1, \psi_1 \vee \psi_2$, $\psi_1 \bigcup \psi_2$, $\Diamond\psi_1$, $\bigcirc\psi_1$.

A.2 Description sémantique

Dans cette section, nous fournirons la sémantique des formules d'état et de chemin, celle des évènements et enfin celle des croyances, désirs et intentions que nous devons à Rao et Georgeff [RG91].

A.2.1 Sémantique des formules d'état et de chemin

Définition 1 (Situation)
Une *situation* ω_t est un monde ω à un moment du temps t.

Définition 2 (Interprétation)
Une *interprétation* M est un tuple, $M = \langle W, E, T, \prec, U, B, G, I, \Phi\rangle$, où

- W est un ensemble de mondes ;
- E est un ensemble d'évènements ;
- T est un ensemble de moments du temps (*time point*) ;
- $\prec$ est une relation d'ordre stricte sur les moments de temps ;
- U est l'univers de discours ;
- Φ est une fonction qui associe des éléments de prédicats du premier ordre à des éléments de U pour chaque monde et moment du temps ;
- $B, G, I \subseteq W \times T \times W$ sont des relations qui associent respectivement à une situation, les mondes de croyances, d'objectifs et d'intentions accessibles. Pour simplifier nous définissons également une relation R qui se réfère aux trois relations ; R_t^w désigne cette relation au temps t et dans le monde w.

Définition 3 (Arbre temporel)
Chaque monde w de W, appelé *arbre temporel*, est un tuple $\langle T_w, A_w, S_w, W_w\rangle$ où

- $T_w \subseteq T$ est l'ensemble des moments du temps du monde w ;
- A_w est identique à $\prec$ restreint au moment du temps de T_w ;
- $S_w : T_w \times T_w \rightarrow E$ est une fonction qui représente l'évènement qui s'est produit avec succès entre deux moments ;

– $F_w : T_w \times T_w \rightarrow E$ est une fonction qui représente l'évènement qui s'est produit sans succès entre deux moments adjacents.

Notons que les domaines de S et de F doivent être disjoints.

Définition 4 (Sous-monde)
Un *sous-monde* est défini comme un sous-arbre d'un monde dont les formules ont la même valeur de vérité. Un monde w' est un sous monde de w, et est noté $w' \sqsubseteq w$ ssi :

1. $T_{w'} \sqsubseteq T_w$
2. $\forall u \in T_w, \Phi(q, w', u) = \Phi(q, w, u)$ où q est symbole de prédicat
3. $\forall u \in T_w, R_u^w = R_u^{w'}$
4. $A_{w'}$ est A_w restreint aux moments de temps de $T_{w'}$, et de même pour $S_{w'}$ $F_{w'}$

Définition 5 (Formule d'état et formule de chemin)
Soit une interprétation M, et une assignation de variable v. v_d^i est la fonction qui associe d à la variable i et qui est identique à v pour toutes les autres valeurs.

La sémantique d'une *formule d'état* est exprimée par :

$M, v, w_t \models q(y_1, \cdots, y_n) ssi (v(y_1), \cdots, v(y_n)) \in \Phi[q, w, t]$ où q est un symbole de prédicat ;
$M, v, w_t \models \neg\phi$ ssi $M, v, w_t \not\models \phi$;
$M, v, w_t \models \phi_1 \bigvee \phi_2$ ssi $m, v, w_t \models \phi_1$ ou $m, v, w_t \models \phi_2$;
$M, v, w_t \models \exists i\phi$ ssi $M, v_d^i, w_t \models \phi$ pour un certain d dans U.

La sémantique d'une *formule de chemin* est exprimée par :

$M, v, (w_{t_0}, w_{t_1}, \cdots) \models \phi$ ssi $M, v, w_t \models \phi$ où ϕ est une formule d'état ;
$M, v, (w_{t_0}, w_{t_1}, \cdots) \models \bigcirc\psi$ ssi $M, v, (w_{t_1}, \cdots) \models \psi$;
$M, v, (w_{t_0}, w_{t_1}, \cdots) \models \Diamond\psi$ ssi $\exists k, k \geq 0$ tel que $M, v, (w_{t_k}, \cdots) \models \psi$;
$M, v, (w_{t_0}, w_{t_1}, \cdots) \models \psi_1 \bigcup \psi_2$ ssi :

$\exists k, k \geq 0$ tel que $M, v, (w_{t_k}, \cdots) \models \psi_2$ et $\forall j, 0 \leq j \leq k, M, v, (w_{t_j}, \cdots) \models \psi_1$, ou
$\forall j \geq 0, M, v, (w_{t_j}, \cdots) \models \psi_1$

Définition 6 (*Optional* et *Inevitable*)
La sémantique des opérateurs *optional()* et *inevitable()* peut être définis par :

$M, v, w_{t_0} \models$ *optional*(ψ) ssi il existe un chemin complet (w_{t_0}, w_{t_1}) tel que $M, v, (w_{t_0}, w_{t_1}, \cdots) \models \psi$.
inevitable(ψ) est défini comme $\neg$*optional*(ψ).
$\Box$ est défini comme $\neg\Diamond\neg\psi$

Une formule bien fondée qui ne contient pas d'occurrence (positive) de l'opérateur *inevitable*() en dehors des parenthèses des opérateurs BEL, GOAL, INTEND, est appelée *O-formule*. L'opérateur qui ne contient pas d'occurrence (positive) de l'opérateur *optional*() est appelé *I-formule*.

A.2.2 Sémantique des évènements

Les évènements transforment un moment du temps en un autre moment. La formule $succeeded(e)$ exprime un succès de l'exécution tandis que $failed(e)$ exprime un échec. $done(e)$ représente le traitement de l'évènement qui est $succeeded(e)$ ou $failed(e)$. Les notions $succeeds$, $fails$ et $does$ sont définies similairement, mais exigent des évènements qui se produisent sur tous les chemins émanant du point auquel la formule est évaluée. La sémantique des évènements se définit comme :

$M, v, w_{t_1} \models succeeded(e)$ ssi $\exists t_0 \in T_w$ tel que $S_w(t_0, t_1) = e$
$M, v, w_{t_1} \models failed(e)$ ssi $\exists t_0 \in T_w$ tel que $F_w(t_0, t_1) = e$
$done(e) \equiv succeeded(e) \vee failed(e)$
$succeed(e) \equiv inevitable \bigcirc (succeeded(e))$
$fails(e) \equiv inevitable \bigcirc (failed(e))$
$does(e) \equiv inevitable \bigcirc (done(e)$

A.2.3 Sémantique des croyances, désirs et intentions

Un agent a une croyance β à un instant t, noté BEL(β) si et seulement si β est vrai dans tous les mondes de croyances accessibles à l'agent au temps t. De la même manière, un agent possède un désir (ou un objectif) γ au temps t, si et seulement si γ est vrai dans tous les mondes de désirs accessibles à l'agent au temps t. Il a une intention ι si et seulement si elle est vraie dans tous les mondes d'intentions qui lui sont accessibles au temps t.

$M, v, w_w \models$ BEL(β) ssi $\forall w' \in B_t^w \ M, v, w'_w \models \beta$
$M, v, w_w \models$ GOAL(γ) ssi $\forall w' \in G_t^w \ M, v, w'_w \models \gamma$
$M, v, w_w \models$ INTEND(ι) ssi $\forall w' \in M_t^w \ M, v, w'_w \models \iota$

A.3 Axiomes

Nous venons de définir la sémantique des concepts du modèle BDI. Dans cette partie, nous présentons des axiomes qui décrivent les relations entre ces concepts.

Compatibilité croyance-désir

Si un agent adopte une O-formule α comme désir, il doit croire en cette formule.

$$\text{GOAL}(\alpha) \supset \text{BEL}(\alpha)$$

Compatibilité désir-intention

Si l'agent adopte une O-formule α comme intention, il doit adopter cette formule comme désir.

$$\text{INTEND}(\alpha) \supset \text{GOAL}(\alpha)$$

Intentions liées aux actions

Si un agent a une intention sur une action e, il effectuera cette action.

$$\text{INTEND}(does(e)) \supset does(e)$$

Croyances concernant des intentions

Si un agent a une intention, il a la conviction qu'il a cette intention.

$$\text{INTEND}(\iota) \supset \text{BEL}(\text{INTEND}(\iota))$$

Croyances concernant des désirs

Si un agent a un désir γ, l'agent a la conviction qu'il doit accomplir ce désir.

$$\text{GOAL}(\gamma) \supset \text{BEL}(\text{GOAL}(\gamma))$$

Désirs concernant des intentions

Si un agent a une intention ι, il a le désir d'accomplir cette intention.

$$\text{INTEND}(\iota) \supset \text{GOAL}(\text{INTEND}(\iota))$$

Consciences des évènements

L'agent a conscience de tous les évènements qui se produisent.

$$done(e) \supset \text{BEL}(done(e))$$

Pas d'ajournement infini d'intentions

Enfin, le dernier axiome exprime qu'un agent ne peut reporter sans cesse une intention. S'il n'arrive pas à l'accomplir, il doit finir par l'abandonner.

$$\text{INTEND}(\iota) \supset inevitable\Diamond(\neg\text{INTEND}(\iota))$$

Annexe B

Description des méthodologies orientées agents

B.1 Tropos

Tropos[1] [BPG+04] est une méthodologie de développement de logiciel créée par un groupe de chercheurs provenant de diverses universités du Canada et d'Italie. Une des différences significatives entre Tropos et les autres méthodologies est sa focalisation sur l'analyse des besoins préalables où les acteurs et leurs intentions sont identifiés et analysés. Le processus de développement de Tropos consiste en cinq phases :

1. l'identification des besoins préalables ou « *early requirements* » ;
2. l'identification des besoins ultérieurs ou « *late requirements* » ;
3. le design de l'architecture ;
4. le design détaillé du système ; et enfin,
5. l'implémentation.

La phase de spécification des besoins est influencée par le *framework* de modélisation i* de Eric Yu's. Tropos utilise les concepts d'acteurs et d'objectifs pour modéliser les intervenants du domaine et leurs intentions respectives. Il s'agit également de modéliser le système cible dans son environnement c'est-à-dire d'étudier la coordination des processus entre ce système et les acteurs humains.

Le design architectural définit l'architecture globale du système en terme de sous-systèmes représentés comme acteurs. Les dépendances entre ces sous-acteurs décrivent les processus de coordination entre les composants du système. La phase suivante dans le processus de développement de Tropos est le design détaillé, où les agents sont spécifiés en terme de capacités et de plans. Un ensemble de diagrammes proposés par l'AUML[2] peut être utilisé. Au final, le design détaillé spécifie les interactions entre les agents.

[1] `http://www.troposproject.org/`

[2] Pour rappel, *Agent Unified Modeling Language*.

Ensuite, vient la dernière phase correspondant à la phase d'implémentation. Tropos choisit une plate-forme BDI[3] à savoir spécifiquement JACK Intelligent Agents[4] pour implémenter les agents. JACK permet d'implémenter les cinq éléments importants que sont : les agents, les capacités, les relations de base de données, les événements et les plans. Nous ne nous attardons volontairement pas trop longtemps sur ce sujet puisque les plates-formes de développement orientées agents constituaient le sujet du chapitre 5. Pour en terminer avec la phase d'implémentation, il s'agit de faire correspondre chaque concept de la phase de design détaillé aux cinq concepts de JACK.

En résumé, la méthodologie **Tropos** peut se targuer d'être complète. Des besoins de base jusqu'au désign détaillé, elle offre des documents de description précis et circonstanciés.

B.2 MaSE

L'objectif de MaSE[5] (*Multiagent Systems Engineering*) est de fournir une méthodologie avec un cycle complet afin d'assister les développeurs dans le design et le développement d'un système multi-agents [DWS01]. Cela donne une description complète du processus qui guide un développeur de la spécification initiale du système à l'implémentation de celui-ci. Ce processus consiste en pas moins de sept étapes, divisées en deux phases.

La phase **Analyse** de MaSE inclut trois étapes du processus général. Premièrement, l'étape de capture des objectifs permet à l'analyste d'identifier les objectifs et leur structure et de les représenter en hiérarchie. La seconde étape consiste à écrire les scénarii principaux ainsi que de construire les diagrammes de séquence. La définition des rôles est la troisième et dernière étape de cette phase, à travers un modèle de rôle et un modèle de tâche concurrente.

La première étape de la phase **Design** est appelée « *Creating Agent Classes* » durant laquelle est construit un diagramme d'agents. Viennent ensuite les diagrammes de communication, deuxième étape, puis la définition de l'architecture des agents et des composants. En termes d'architecture agent, MaSE n'impose pas de plate-forme d'implémentation particulière. La quatrième et dernière étape est celle du design du système.
MaSE est accompagnée d'un outil de support sous la forme d'un « agentTool » que trop peu de méthodologies peuvent se vanter de posséder. Sa dernière version implémente les sept étapes de MaSE.

En résumé, MaSE est complète et les documents construits sur base de celle-ci sont d'un très bon niveau de détail. Elle considère les agents comme des simples

[3] Pour rappel, *Belief-Desire-Intention*.
[4] `http://www.agent-software.com`
[5] `http://www.cis.ksu.edu/~sdeloach/ai/projects/mase.htm`

processus logiciel qui interagissent entre eux pour atteindre un objectif général.

B.3 Prometheus

Développée pendant plusieurs années par Lin Padgham et Michael Winikoff [WP04] en collaboration avec Agent Oriented Software Pty. Ltd., une compagnié commerciale australienne, la méthodologie *Prometheus* [6] est une méthodologie AOSE[7] destinée principalement à des non-experts [PW02]. Elle a été enseignée avec succès à des étudiants et a fait son entrée dans les entreprises. Elle consiste en trois phases : **spécification du système**, **design de l'architecture** et **design détaillé**.

La première phase implique deux activités : *déterminer l'environnement du système* et *déterminer les objectifs et fonctionnalités du système*. Perceptions, actions, scénarii d'utilisation[8] seront définis dans des descripteurs.

La seconde phase implique quant à elle trois activités : *la définition des types d'agents, le design de la structure globale du système et la définition des interactions entre les agents*. Cette phase aboutit au diagramme de vue d'ensemble du système qui représente la structure du système et qui est considéré comme l'*artifact*[9] le plus important dans Prometheus.

La troisième et dernière phase aborde *le système interne des agents* et déduit *comment ceux-ci accomplissent leurs tâches* dans le système. Capacités, événements internes aux agents, plans et structures de données détaillées seront étudiés pour chaque type d'agent.

La méthodologie Prometheus est accompagnée de deux outils :

- **L'environnement de développement JACK (JDE[10])** développé par Agent Oriented Software et qui inclut un outil de design qui permet de dessiner des diagrammes de vue d'ensemble du système ou *« overview diagrams »*. Ces diagrammes sont liés avec le modèle sous-jacent ce qui signifie que des changements effectués sur les diagrammes, par exemple l'ajout d'un lien entre un plan et un événement, sont directement répercutés dans le modèle et dans le code JACK correspondant.
- **L'outil de design de Prometheus (PDT[11])** fournit des formulaires pour encoder les designs des entités. Il s'occupe également des vérifications croisées[12] afin de s'assurer de la consistance des données et génère un document de design ainsi que des diagrammes de vue d'ensemble.

[6] `http://www.cs.rmit.edu.au/agents/`
[7] *Agent Oriented Software Engineering*
[8] *Use cases* en anglais.
[9] Pour rappel, il s'agit du nom souvent donné au document d'analyse d'une méthodologie de développement logiciel.
[10] *JACK Development Environment*
[11] *Prometheus Design Tool*
[12] Plus connues sous le nom de *Cross-Checking* en anglais

B.4 Gaia

Gaia est le nom d'une hypothèse avancée par l'écologiste James Lovelock et qui affirme que « *tous les organismes vivants de la terre peuvent être vus comme des composants d'une entité unique qui régule l'environnement de la Terre* ». Le thème des nombreuses entités hétérogènes agissant ensemble pour atteindre un objectif unique est le thème central de recherche à propos des systèmes multi-agents et est une des considérations clés dans la construction de cette méthodologie [WJK00]. **Gaia** [WJK00] a été développée durant de longues années par des personnes expérimentées dans la construction de systèmes multi-agents. Elle est à la fois *générale* en ce sens qu'elle est applicable à un large éventail de systèmes multi-agents et *complète* car elle prend en charge les aspects macro-sociétaux mais également les aspects micro-agents des systèmes. Gaia est fondée sur la vue d'un système multi-agents comme une organisation de calcul qui consiste en des rôles d'interaction variés.

En appliquant Gaia, les concepteurs d'un système se déplacent de concepts abstraits à des concepts de plus en plus concrets. Gaia est composé de trois phases : *spécification des besoins*, *analyse* et *design*. Plus généralement, ces trois phases peuvent être vues comme des étapes d'un processus de développement qui consiste à construire des modèles de plus en plus détaillés du système. Systématiquement, l'objectif est de passer d'une définition des besoins à un design suffisamment détaillé pour être implémenté directement. Les modèles principaux et leurs relations sont repris dans la figure B.1 ci-dessous.

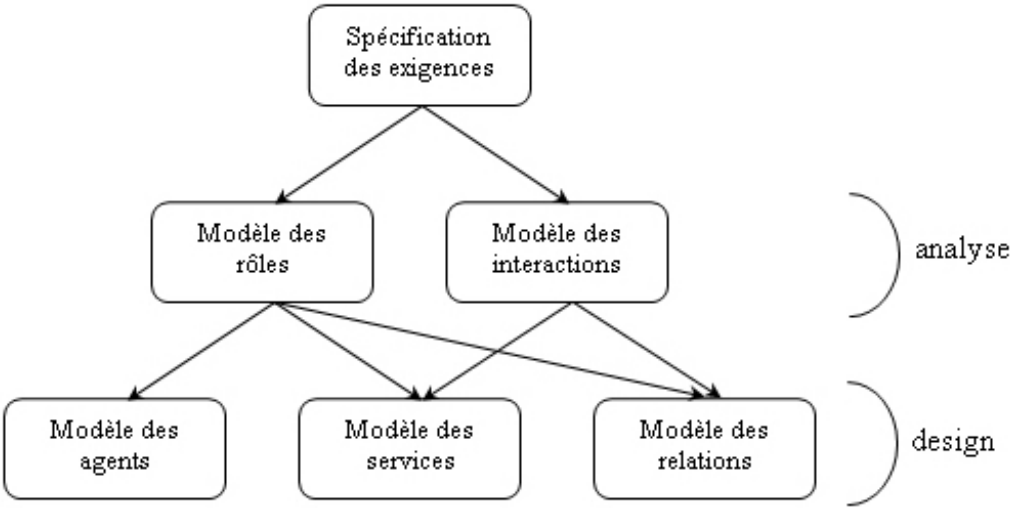


FIG. B.1 – Les relations entre les modèles de Gaia [WJK00].

Gaia emprunte quelques terminologies et notations directement aux analyses et designs orientés objets. Cependant, ce n'est pas une tentative naïve d'appliquer de telles méthodes au développement orienté agents. Au contraire, Gaia fournit un ensemble spécifique de concepts agents à travers lequel un ingénieur logiciel peut

comprendre et modéliser un système complexe. En particulier, Gaia encourage un développeur à penser la construction d'un système multi-agents comme un processus de *design organisationnel*.

L'objectif de la phase d'*analyse* est de comprendre le système et sa structure (sans aucune référence à tout détail d'implémentation). Comme nous l'avons déjà évoqué, il s'agit de voir le système comme une organisation et d'étudier ses différentes caractéristiques comme les rôles et permissions associées, les relations entre les entités de l'organisation, les responsabilités, etc.

Ensuite, l'objectif d'un processus classique de *design* est de transformer les modèles abstraits dérivés pendant l'étape d'analyse en modèles de niveau d'abstraction suffisamment bas pour qu'ils puissent être implémentés facilement. Ce n'est pas le cas avec un design orienté agents. A contrario, l'objectif dans Gaia est de transformer les modèles analytiques dans un niveau d'abstraction suffisamment bas pour que des techniques de design traditionnelles (en incluant les techniques orientées objets) puissent être appliquées afin d'implémenter les agents. En d'autres mots, Gaia s'occupe de *la manière dont les agents d'une même société coopèrent* pour réaliser les objectifs du système et ce qu'il est requis de chacun des agents. Par contre, *la façon dont un agent réalise ses services* est au-delà du champ analytique de Gaia. Nous pouvons d'ores et déjà parler d'un manque de processus de design détaillé — intentionnellement absent suite au désir de généralité — qui pourrait fortement nous handicaper dans la réalisation de notre premier système multi-agents.

Puisque Gaia ne fait pas partie de la comparaison sur laquelle nous nous basons, continuons de dresser quelques traits supplémentaires de cette méthodologie de développement de logiciel orientée agents. En tant que méthodologie AOSE, Gaia est plutôt destinée à des applications de grandes échelles avec les caractéristiques suivantes :

- les agents sont des systèmes de calcul, faisant chacun utilisation de ressources de calcul ;
- les agents sont hétérogènes et peuvent être implémentés en utilisant des langages de programmation, des architectures et des techniques différents ;
- la structure d'organisation du système est statique en ce que les relations inter-agents ne changent pas à l'exécution ;
- les capacités des agents et les services qu'ils fournissent sont statiques c'est-à-dire que eux non plus ne changent pas à l'exécution ;
- le système global contient un nombre relativement faible de types d'agents différents (moins de 100).

Annexe C

Les technologies à l'origine du Workflow Management

C.1 La bureautique

Nul doute que la bureautique peut être considérée comme **la principale origine du workflow management** [EN80]. Ce n'est pas l'automatisation des tâches individuelles qui est importante mais bien **le contrôle et la conduite de l'exécution des processus business**. Giampio Bracchi et Barbara Pernici [BP84] ont dressé une liste des besoins associés aux systèmes d'informations bureautiques et qui peuvent être appliqués aux systèmes de WM. Parmi ceux-ci, nous trouvons :

- les activités de planning ;
- l'intégration de la fonction ;
- l'assistant personnel ;
- le management des tâches.

C.2 La gestion de base de données

Les systèmes de gestion de bases de données conventionnels sont passifs en ce sens qu'ils ne font que manipuler des données en réponse à des requêtes explicites d'utilisateurs ou d'applications. Cette divergence entre le support disponible de ces systèmes et les besoins des applications à longue exécution a mené au développement de systèmes de gestion de bases de données actifs [DHW].

La principale idée de ces systèmes actifs est d'enrichir les systèmes passifs avec des règles *évènement-condition-action* permettant d'implémenter des activités multi-étapes qui peuvent être considérées comme de **simples workflows**.

C.3 E-mail

Certaines caractéristiques du workflow management peuvent être facilement retrouvées dans les systèmes de courrier électronique. Teamroute de Digital Equipment Cooperation[1] peut être considéré comme un précurseur des systèmes de workflow management. En plus des caractéristiques habituelles (*header* et *body*), les courriers électroniques contiennent des **informations de routage** du courrier (liste des adresses qui doivent recevoir le mail en fonction de certains critères). Cette fonctionnalité est essentielle pour le workflow management.

C.4 Management de document

L'avènement de l'ordinateur dans les bureaux a mené au remplacement des documents en papier par les documents électroniques. En plus de la mise à disposition des documents aux utilisateurs, fonctionnalité offerte par les systèmes passifs, les systèmes actifs de management de document incorporent des services et fonctions qui sont basés sur le temps (par exemple, un document peut-être présenté pour une relecture après une certaine période de temps). Bien que simplifié à l'excès, il s'agit d'une **première étape vers le workflow management**.

C.5 Le management des processus logiciel

Un processus logiciel guide et assiste les personnes impliquées dans des processus de développement de logiciel. Par exemple, un processus software coordonne le développement distribué de larges packages software de telle façon que les designs partiels produisent des pièces consistantes du software. Certaines approches du management de processus software [DG94] montrent des **concepts et des techniques** qui sont aussi essentielles pour le workflow management.

C.6 La modélisation des processus business

La reconception des processus business fait beaucoup parler de lui de nos jours. Actuellement, les organisations traversent une période qui tient de la crise de « mivie », en se questionnant et repensant leurs façons de pratiquer le business. Lors de restructuration, nombreuses sont les compagnies qui réalisent que les processus et les technologies en place ne sont plus optimales.

La modélisation des processus business insiste sur l'accomplissement du travail et non sur l'exécution de tâches singulières. Le point de vue est très large et l'étude porte sur différents aspects du business [Kat90] : finance, distribution, ligne de production, besoin en informations et données, flux d'informations et données à travers les processus business, etc.

[1] En 1998, Compaq Computer annonça l'acquisition de Digital Equipment Cooperation.

Les idées conceptuelles de la modélisation, spécifiquement son **approche large** et **multi-aspects**, induisent le développement de software de workflow management.

C.7 La modélisation d'entreprise et architecture

Nous avons déjà mentionné au point 6.3.1 (page 71) qu'un important prérequis pour le workflow management est sa vue du travail centrée *processus*. Dans le cas de la modélisation d'entreprise, une telle vue centrée processus est obligatoire. Le WM apprend de la modélisation d'entreprise à **prendre en compte tous les aspects** d'un système.

Annexe D

La notation AUML

D.1 Introduction

Ce chapitre fournit une brève description des notations AUML[1], qui sont utilisées par la méthodologie Prometheus. AUML est une version révisée de UML. Elle est mieux adaptée à l'élaboration d'agents.

La notation AUML est une extension des diagrammes d'interactions UML. Un schéma AUML possède une ligne du temps pour chacun des agents ainsi que des messages représentés par des flèches reliant les lignes du temps de deux agents. Il est inséré dans un cadre contenant en haut à gauche «sd» pour «sequence diagram» suivi du nom du protocole représenté par le diagramme d'interactions. Un exemple simple, présenté à la figure D.1, décrit un agent *utilisateur* qui envoie un message de *requête* à un agent *système*. Ce dernier lui renvoie ensuite une *réponse*.

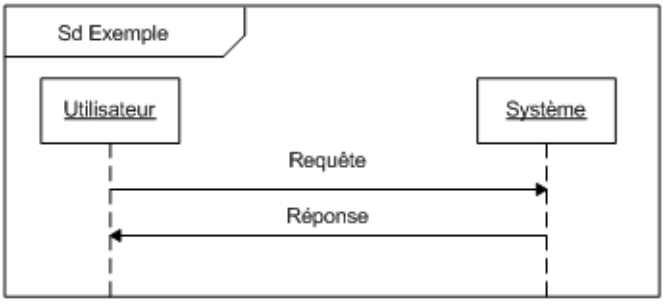


FIG. D.1 – Exemple simple de protocole sous forme de diagramme AUML

Le formalise AUML autorise également des alternatives, du parallélisme, des options, des boucles, ... qui sont spécifiés dans des boîtes. Une boîte (*box*) est une partie du schéma qui contient des messages et d'autres boîtes. Chaque boîte possède une étiquette qui décrit le type de boîte. Les différents types de boîtes sont décrits dans la section suivante. Une boîte peut spécifier un garde (*guard*) qui est exprimé sous la forme de texte entre crochets et qui exprime une condition

[1] `www.auml.org`

qui doit être vraie pour que le contenu de la boîte soit exécuté. Certaines boîtes peuvent être divisées en plusieurs parties, appelées régions et séparées par une ligne composée de tirets.

D.2 Les types de boîtes

Les types de boîtes du formalisme AUML sont définis dans les paragraphes suivants.

Alternative Une boîte alternative est divisée en plusieurs régions. Une seule partie peut être exécutée. Cette boîte doit comporter un ou des gardes. La dernière région peut contenir le garde «else».

Option Ce type de boîte est composé d'une seule région. Elle doit contenir un garde qui spécifie quand l'option doit être exécutée. Une boîte option est équivalente à une boîte alternative, composée de deux régions, dont la seconde serait vide.

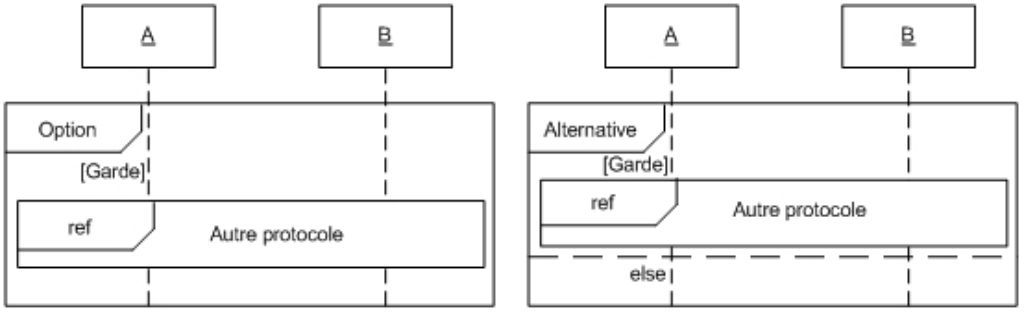


FIG. D.2 – Boîte option et boîte alternative équivalente

Break La boîte est séparée en plusieurs régions. Au maximum, une des régions peut être exécutée. Une boîte break est équivalente à une boîte alternative composée de plusieurs régions, dont la dernière est vide.

Parallel Il s'agit d'une boîte composée de plusieurs régions, où chacune des régions s'exécute simultanément. La séquence des messages est définie par une séquence de lettres.

Loop Cette boîte ne peut contenir qu'une seule région. Elle signifie que la région est répétée un certain nombre de fois. Le nombre de répétition est précisé dans l'étiquette.

Ref Ce type de boîte est un peu différent des autres dans le sens où il ne peut pas contenir d'autres boîtes. Cette boîte contient le nom d'un autre protocole. Il s'agit d'une forme d'appel. La boîte fait donc référence à un autre protocole.

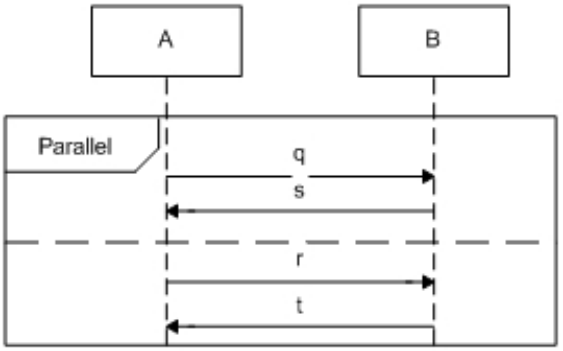


FIG. D.3 – Boîte de type « parallel »

D.3 Les labels de continuations

Les boîtes sont des mécanismes qui permettent de décrire des flux de contrôle. Il existe un autre mécanisme dit de « continuation ». Il est composé de continuations entrantes (appelée *label*) et sortantes (*goto*). Un *goto* est représenté par une flèche sortante vers la droite. Un *label* est composé d'une flèche entrante provenant de la gauche. Quand une interaction atteint un *goto*, elle se poursuit à partir du *label* du même nom. Bien entendu, chaque *goto* ne peut correspondre qu'à un seul *label*. Par contre, il peut exister plusieurs *goto* renvoyant un même *label*.

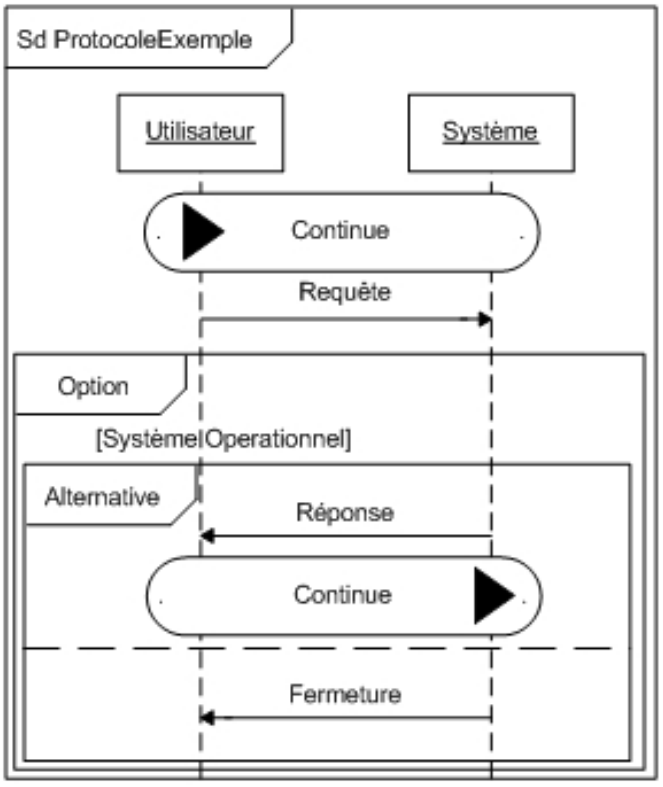


FIG. D.4 – Exemple de protocole avec une continuation

Annexe E

Schémas ERA des croyances

E.1 Première itération

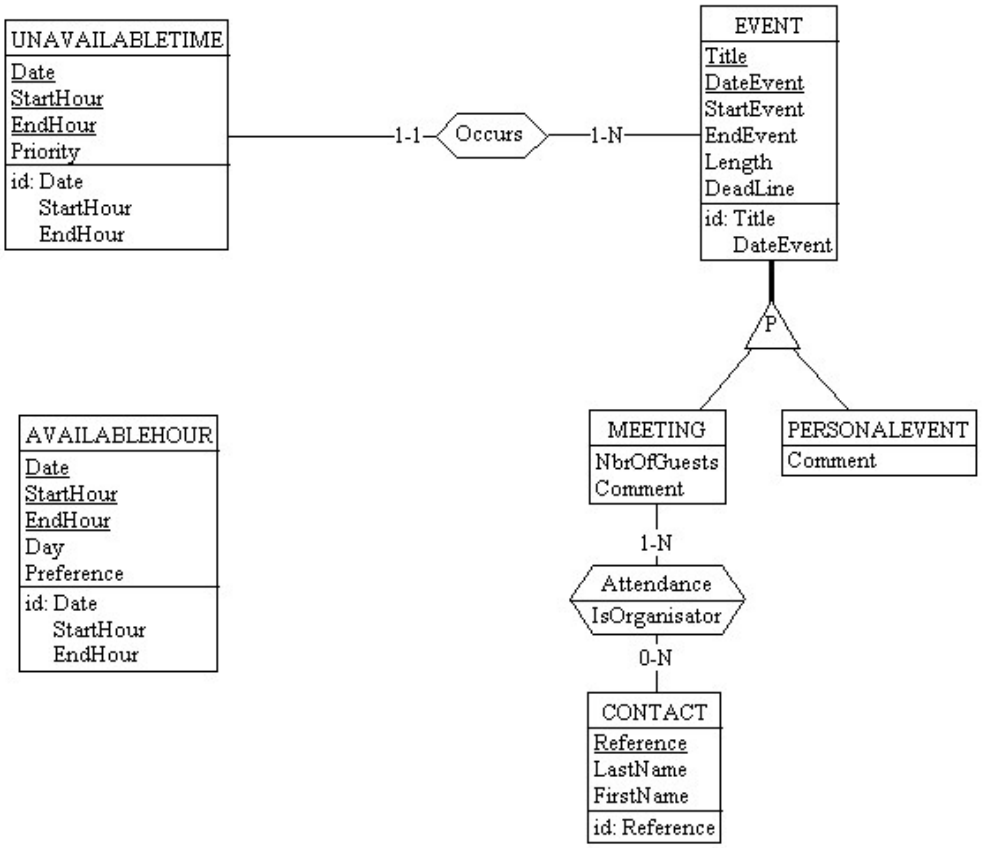


FIG. E.1 – Schéma ERA représentant les croyances de l'agent `Diary`

E.2 Deuxième itération

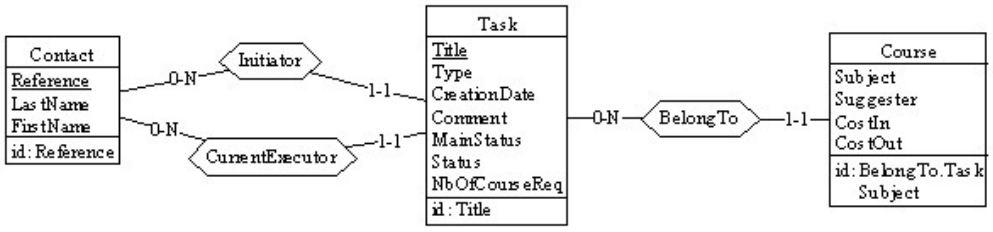


FIG. E.2 – Schéma ERA représentant les croyances de l'agent `Task`

Annexe F

Schémas Jack Design Tool

F.1 Introduction

L'environnement de développement JDE (pour *Jack Development Environment*) de la plate-forme JACK, développée par Agent Oriented Software, inclut un outil de design qui permet de dessiner des diagrammes de vue d'ensemble du système ou « *overview diagrams* ». Ces diagrammes sont liés avec le modèle sous-jacent ce qui signifie que des changements effectués sur les diagrammes, par exemple l'ajout d'un lien entre un plan et un événement, sont directement répercutés dans le modèle et dans le code JACK correspondant.

Dans cette annexe, nous dévoilons les schémas que nous avons réalisés durant notre projet à la « University of Technology, Sydney ». Ceux-ci sont liés à l'étude de cas présentée dans les chapitres10 et 11.

FIG. F.1 – Légende de l'outil de design de Jack

F.2 Première itération

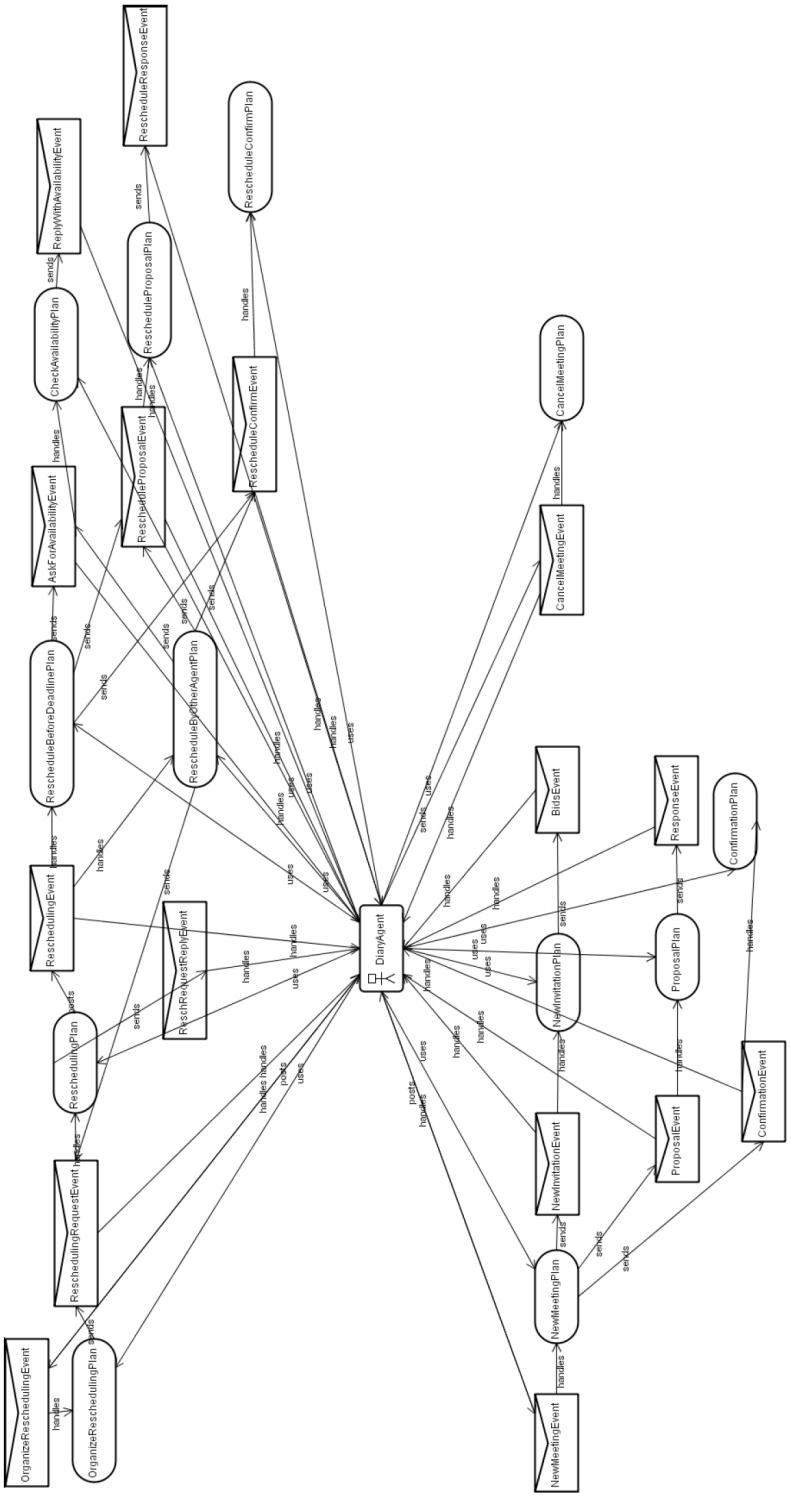


FIG. F.2 – Agent `Diary`.

F.3 Deuxième itération

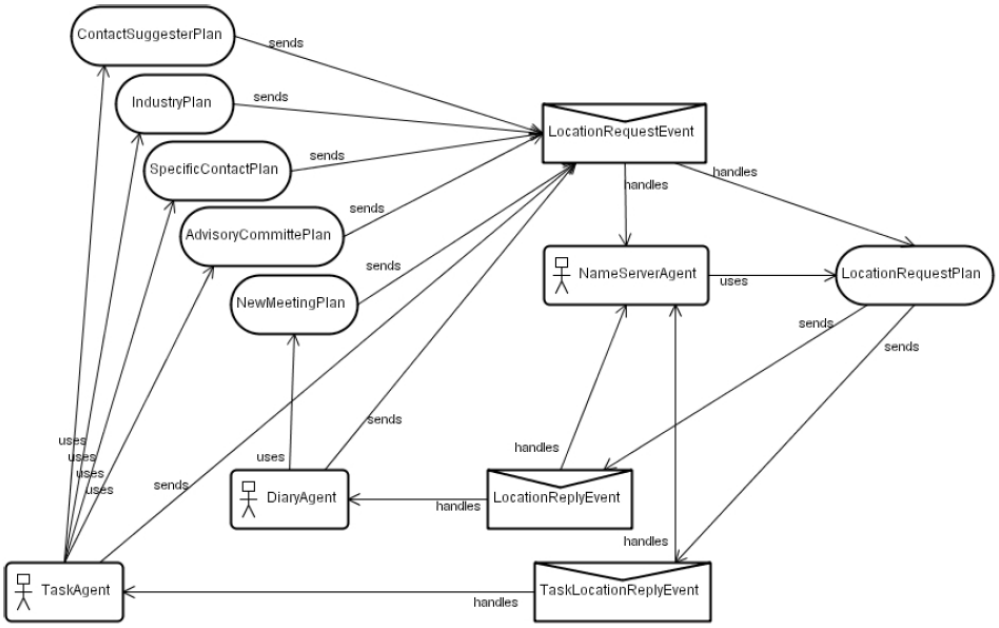


FIG. F.3 – Agent `NameServer`.

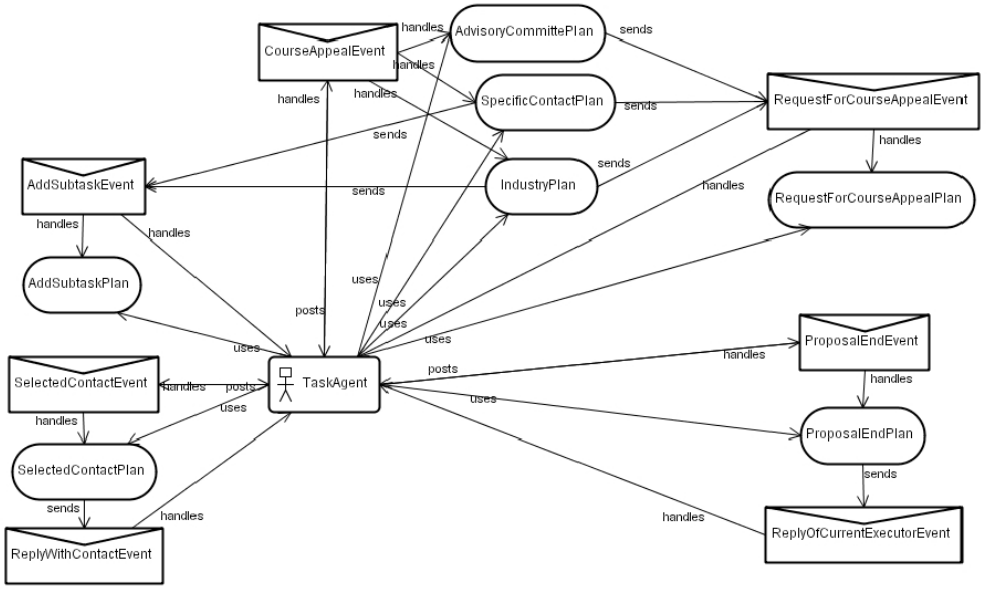


FIG. F.4 – Fonctionnalité *Appel à cours*.

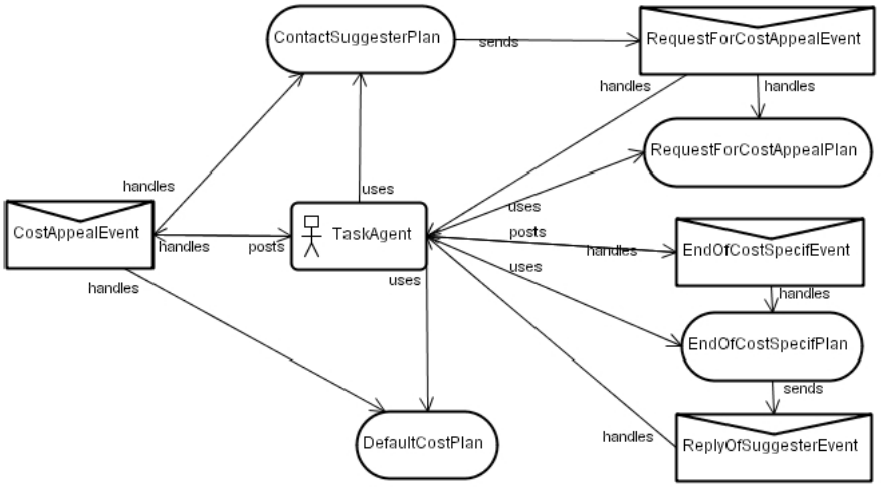


FIG. F.5 – Fonctionnalité *Evaluation des coûts.*

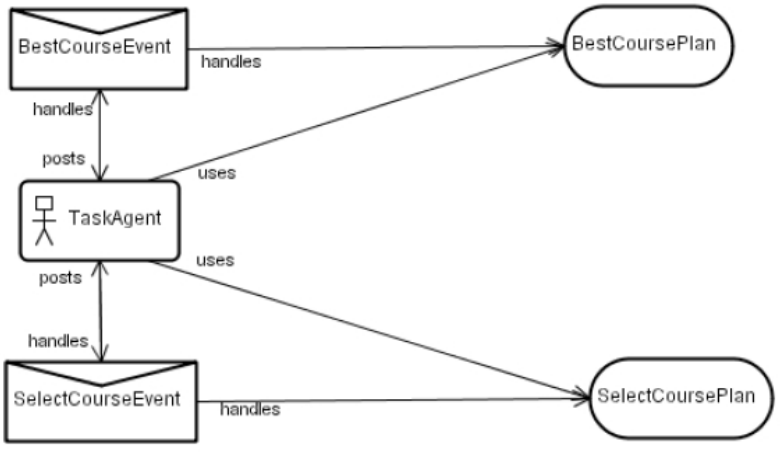


FIG. F.6 – Fonctionnalité *Sélection du meilleur cours.*

Annexe G

Le déploiement du système

G.1 Déploiement du système

La figure G.1 nous montre un exemple de déploiement du système. Dans cet exemple, il faut séparer les cinq acteurs *Professeur X, Contact Comité Consultatif, Professeur Y, Professeur Z* et *Contact Industrie* de la station de travail restante.

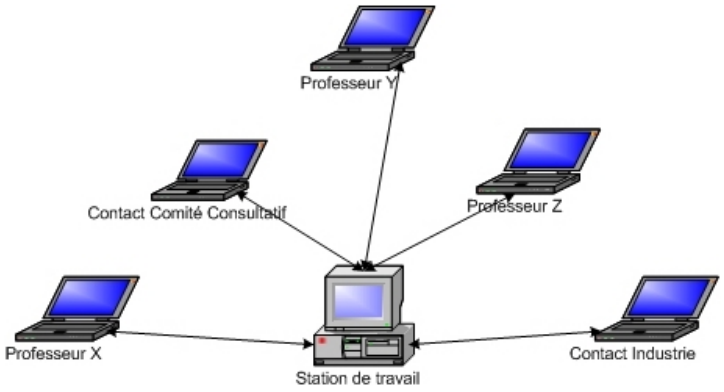


FIG. G.1 – Illustration d'un exemple de déploiement.

Sur chacune des machines de ces acteurs (figure G.2), un processus est démarré et gère l'exécution de deux agents (un `DiaryAgent` et un `TaskAgent`) ainsi que d'un contrôleur qui s'occupe de l'interface homme-machine entre les deux agents et l'utilisateur. Sur une dernière machine, un processus est démarré pour exécuter l'agent `NameServer` qui sert de serveur de noms pour les autres agents : lorsqu'un agent désire connaître l'agent qui est en charge d'un utilisateur devant par exemple être invité à une réunion, il peut contacter le serveur de noms qui est capable de lui donner l'identifiant et la localisation de l'agent correspondant sous la forme *agent@portal*. Cette forme de localisation sera expliquée à la section suivante.

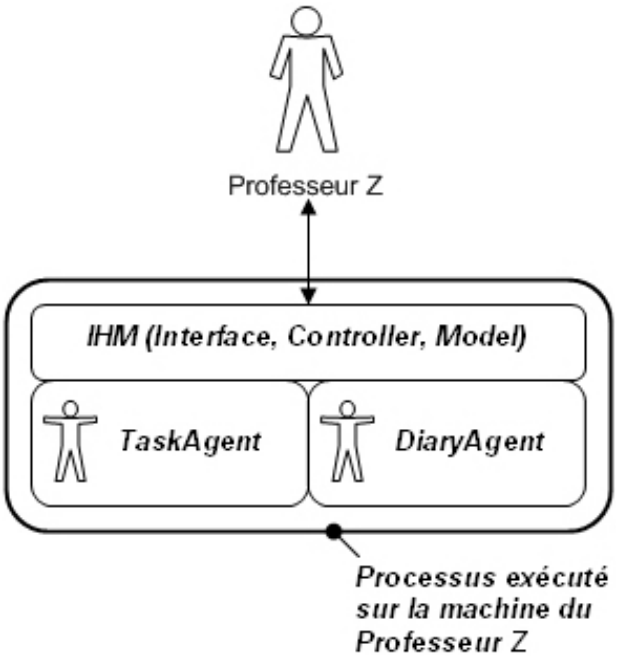


FIG. G.2 – Processus exécuté sur la machine d'un acteur.

G.2 Configuration de la communication inter-agents

Dans le cas d'une communication locale (les agents s'exécutent au sein d'un même processus d'un système d'exploitation), le routage des messages inter-agents est trivial. L'agent émetteur requiert uniquement le nom de l'agent destinataire (sous Jack, chaque agent possède un nom identifiant) pour pouvoir lui envoyer un message.

Dans notre cas de figure, les agents ne s'exécutent pas au sein d'un même processus mais bien dans des processus différents sur des machines différentes. Dans ce cas, la couche de communication de Jack doit être utilisée pour permettre aux agents de communiquer. Cette couche de communication est connue sous le nom de réseau *dci*. Le réseau *dci* est divisé en couches de telle façon que différents mécanismes de transport peuvent être envisagés. Par défaut, Jack utilise le protocole de transport UDP disponible sur les réseaux TCP/IP. UDP est un protocole de transport extrêmement simple et léger, doté d'un modèle de services minimum [KR03]. Contrairement à TCP, il s'agit d'un service sans connexion. De plus, le service de transfert de données procuré par UDP est dit non fiable, c'est pourquoi Jack fournit une fine couche par-dessus UDP qui rend les communications inter-agents fiables.

Quand des agents s'exécutant dans des processus différents ont besoin de communiquer, les processus correspondants doivent savoir comment ils peuvent le faire. Chaque processus a son propre portail[1] grâce auquel il va pouvoir communiquer avec les autres processus. Une paire de portails peut être connectée

[1] *Portal* en anglais.

explicitement ou un serveur de noms peut être utilisé de sorte que les portails sont en mesure de se connecter à celui-ci « on the fly[2] ».

Le nom d'un agent prend la forme suivante : `agent@portal` où `agent` se réfère au nom donné à l'agent lors de sa création[3] et `portal` se réfère au portail du processus dans lequel s'exécute l'agent. La façon la plus simple de configurer un réseau *dci* est de désigner un processus comme serveur de noms. Dans notre cas, c'est le processus correspondant à l'agent `NameServer` qui remplira cette tâche, les processus des autres agents venant enregistrer leur présence auprès de lui.

La configuration devient donc la suivante :

- Sur la machine où s'exécute le processus de l'agent `NameServer`, il s'agit d'initier la couche de communication de Jack de la façon suivante :

```
args = aos.jack.Kernel.init(args)
```

avec,

```
args[0] = "-dci.ns :localhost :3001"
args[1] = "-dci.new :NSAgent=localhost :3001"
```

Ces instructions permettent d'ouvrir un portail pour un serveur de noms sur la machine locale (*localhost*), sur le port 3001. L'adresse IP de cette machine est par exemple 192.128.0.10

- Sur chacune des machines correspondant à un agent X, l'initialisation s'effectue différemment :

```
args = aos.jack.Kernel.init(args)
```

avec,

```
args[0] = "-dci.ns :192.128.0.10 :3001"
args[1] = "-dci.new :AgentX"
```

Ces instructions permettent de signaler la localisation du serveur de noms (*192.128.0.10 :3001*, l'adresse IP de la machine et le port ouvert) et le portail (*AgentX*) qui identifiera, à l'avenir, l'agent *AgentX* qui va être démarré.

[2] Dès que le serveur de noms est démarré, les portails correspondant aux processus des différents agents peuvent se connecter par la suite quand ils désirent faire partie du système.

[3] Instanciation d'un objet de la classe d'agents correspondante.

G.3 Lancement du système

Le lancement du système s'effectue de la façon suivante :

1. Sur une machine du réseau (de préférence pas une machine occupée par un utilisateur de l'application), exécuter le fichier `ns.bat` qui contient les commandes d'initialisation de la couche de communication de Jack et les commandes de lancement de l'agent `NameServer`, le serveur de noms.
2. Sur chacune des machines des utilisateurs, exécuter au sein d'un processus le fichier `AgentsX.bat`[4] correspondant. Ce fichier contient les commandes d'initialisation de la couche de communication de Jack, l'enregistrement des agents auprès du serveur de noms et le lancement de l'application à proprement dite : le contrôleur, l'interface graphique, le modèle et les deux agents (`DiaryAgent` et `TaskAgent`).

[4] Par exemple, Agents1.bat pour l'utilisateur 1.